THE MULTICS COMMAND REPERTOIRE

The following facilities are the ones considered to be part of Multics and are described in this manual. Detailed specifications on each of the commands mentioned below can be found, filed in alphabetical order by command name, in the MPM Reference Guide section, Commands. In addition, many commands have on-line descriptions that can be obtained with the help command.

The active functions available on Multics are described separately, by functional grouping, in several MPM Reference Guide write-ups: Alphabetical List of Active Functions, Logical Active Functions, Arithmatic Active Functions, Character String Active Functions, Segment Name Active Functions, Data and Time Active Functions, Question Asking Active Functions, and User Paremeter Active Functions. Active functions can be used in a command line in the manner described in the MPM Reference Guide section, The Command Language, under the heading Active Strings.

The user should also consult the list of items in the Author Maintained and/or Installation Maintained Library at his installation, since local language translators and other commands can substantially extend the standard command repertoire. Documentation of the Author Maintained and/or Installation Maintained Library is supplied by the local installation.

The command repertoire is organized by function into the following groups:

Storage System, Creation and Editing of Segments
Storage System, Segment Manipulation
Storage Segment, Directory Manipulation
Storage System, Access Control
Storage System, Formatted Output Facilities
Storage System, Address Space Control
Language Translators, Compilers, Assemblers and Interpreters
Object Segment Manipulation
Debugging and Performance Monitoring
I/O System Control
Command Typing and Control
Communication Among Users
Accounting
Control of Absentee Computations
Miscellaneous Tools

© Copyright, 1973, Massachusetts Institute of Technology
and Honeywell Information Systems Inc.

CBI 61 box 2

1) Storage System, Creation and Editing of Segments

adjust_bit_count	sets bit count of a segment to last nonzero character
basic_system	editor and runner of BASIC programs
compare_ascii	compares supposedly identical ASCII segments, reporting differences
create	creates an empty segment
edm	inexpensive, easy to learn editor
indent	indents a PL/I source segment to make it more readable
qedx	more sophisticated editor
sort_file	sorts ASCII segments line by line

2) Storage System, Segment Manipulation

addname	adds a name to a segment, directory, or link
adjust_bit_count	sets bit count of a segment to last nonzero character
archive	packs segments together to save physical storage breakage
archive_sort	sorts the contents of an archive segment by component segment name
compare	compares segments word by word, reporting differences
compare_ascii	compares supposedly identical ASCII segments, reporting differences
copy	copies a segment
create	creates an empty segment
delete	deletes a segment, but questions if segment is read only
deleteforce	deletes a segment without question
deletename	removes a name from a segment, directory, or link
file_output	directs typewriter output to a segment
fs_chname	an extra powerful rename command
link	creates a directory link to another segment
list, listnames, listotals	prints directory contents
move	moves segment to another directory
names	moves or copies names from one storage system entry to another
rename	renames a segment, link, or directory

© Copyright, 1973, Massachusetts Institute of Technology and Honeywell Information Systems Inc.

reorder_archive	rearrages order of an archive segment
safety_sw_off	turns safety switch off for a segment or directory
safety_sw_on	turns safety switch on for a segment or directory
set_bit_count	sets a given bit count in a segment
status	prints everything known about an entry in a directory
truncate	used to truncate a segment to a specified length
unlink	removes a directory link

3) Storage System, Directory Manipulation

change_default_wdir	sets the default working directory
change_wdir	changes to a new working directory
createdir	creates a directory
delete_dir	destroys a directory
list, listnames, listotals	prints directory contents
print_default_wdir	prints default working directory name
print_wdir	prints current working directory name

4) Storage System, Access Control

delete_iacl_dir	removes an Initial ACL for new directories
delete_iacl_seg	removes an Initial ACL for new segments
deleteacl	removes an access control list (ACL) entry
deletecacl	removes a common access control list (CACL) entry
list_iacl_dir	prints an Initial ACL for new directories
list_iacl_seg	prints an Initial ACL for new segments
listacl	prints an ACL entry
listcacl	prints a CACL entry
set_iacl_dir	adds (or changes) an Initial ACL for new directories
set_iacl_seg	adds (or changes) an Initial ACL for new segments

© Copyright, 1973, Massachusetts Institute of Technology
and Honeywell Information Systems Inc.

setacl	adds (or changes) an ACL entry
setcacl	adds (or changes) a CACL entry

5) Storage System, Formatted Output Facilities

dprint	adds segment to the high speed line printer queue
dpunch	adds a segment to the card punch queue
dump_segment	selective octal dump of segment contents
mail	prints or sends mail
memo	allows users to set reminders for later printout
print	prints any ASCII segment
print_motd	prints the system's message of the day
runoff	formats a text segment according to internal control words
runoff_abs	invokes the runoff command as an absentee job

6) Storage System, Address Space Control

change_wdir	controls directory assumed when relative path names are given
initiate	adds a segment to the address space of a process
list_ref_names	prints all names by which segment is known to a process
print_search_rules	prints path of search for missing segments
print_wdir	prints name of current working directory
set_search_dirs set_search_rules	controls path of search for missing segments
terminate	removes a segment from process address space
where	prints full path name of a segment

7) Language Translators, Compilers, Assemblers, and Interpreters

alm	assembly language for Multics
alm_abs	invokes the ALM assembler as an absentee job
apl	invokes the APL interpreter
basic	translates and optionally runs Version 6 BASIC programs

© Copyright, 1973, Massachusetts Institute of Technology
and Honeywell Information Systems Inc.

basic_run	runs BASIC programs
basic_system	editor and runner of BASIC programs
fortran	standard FORTRAN IV compiler
fortran_abs	invokes the FORTRAN compiler as an absentee job
lisp	a LISP 1.5 interpreter and compiler
lisp_compiler	invokes a LISP compiler
pl1	the Multics PL/I compiler
pl1_abs	invokes the PL/I compiler as an absentee job
qedx	sophisticated editor, with macro facilities (a minor interpreter)
v2pl1_abs	invokes the Version II PL/I compiler as an absentee job
v5basic	translates and optionally runs Version 5 BASIC programs

8) Object Segment Manipulation

bind	packs two or more object segments into a single segment
display_component_name	prints name of bound component, gives offset
print_bind_map	prints information about a bound segment
print_link_info	prints list of entries and outbound links of an object segment

9) Debugging and Performance Monitoring

change_error_mode	adjusts length and contents of status messages
debug	symbolic source language debugger
dump_segment	selective octal dump of segment contents
hold	saves the stack for debugging or later restart
how_many_users	tells how many users are logged in
page_trace	prints list of pages recently demanded
print_linkage_usage	prints map of all current linkage
profile	prints information about execution of individual statement within a program
progress	prints information about the progress of a command as it is being executed

© Copyright, 1973, Massachusetts Institute of Technology
and Honeywell Information Systems Inc.

ready	prints a summary line of CPU time and paging usage
ready_off	suppresses the ready line following commands
ready_on	restores the ready line
reprint_error	allows retyping of earlier status messages
trace_stack	prints stack history
where	prints full path name of a segment

10) I/O System Control

cancel_daemon_request	cancels a previosuly submitted daemon request
close_file	closes open FORTRAN and PL/I files
console_output	restores typewriter output to the typewriter
dprint	adds a segment to the high speed printer queue
dpunch	adds a segment to the card punch queue
file_output	directs typewriter output to a segment
iocall	allows direct calls to I/O system entries
iomode	sets typewriter character conversion modes
line_length	sets typewriter carriage length
list_daemon_requests	prints list of daemon requests currently queued
print_attach_table	prints list of current I/O system stream attachments

11) Command Typing and Control

abbrev	allows user-specified abbreviations for command lines or parts of command lines
answer	answers questions normally asked of the user
do	executes a command line with arguments inserted
enter	enters an anonymous user into the system
exec_com	allows a segment to be treated as a list of commands to be executed
get_com_line, set_com_line	adjusts size of command line buffers

© Copyright, 1973, Massachusetts Institute of Technology and Honeywell Information Systems Inc.

hold	saves the stack for debugging following an accident
login	enters user into the system
logout	exits user from the system
new_proc	creates a new process with a fresh address space
program_interrupt	signals a condition following an accident
release	releases a saved stack
start	restarts a computation following a quit or unexpected signal
walk_subtree	repeats a command in all directories below a given directory

12) Communication Among Users

change_wdir	changes base of operation to another directory
check_info_segs	checks information (and other) segments for changes
help	prints special information segments
link	inserts a directory link to another segment
mail	sends an ASCII message to another user
make_peruse_text	prepares segments for use by peruse_text
peruse_text	selectively prints structural information segments
set_search_dirs	sets path of search for missing segments
unlink	removes a directory link
where	prints full path name of a segment
who	prints list of users currently logged in

13) Accounting

getquota	prints secondary storage quota and usage
movequota	moves secondary storage quota to another directory
resource_usage	prints resource consumption for the month

Copyright, 1973, Massachusetts Institute of Technology
and Honeywell Information Systems Inc.

14) Control of Absentee Computations

alm_abs	invokes the ALM assembler as an absentee job
cancel_abs_request	cancels a previously submitted absentee job request
enter_abs_request	adds a request to the absentee job queue
fortran_abs	invokes the FORTRAN compiler as an absentee job
list_abs_request	prints list of absentee job requests currently queued
pl1_abs	invokes the Version I PL/I compiler as an absentee job
runoff_abs	invokes the runoff command as an absentee job
who	prints list of absentee jobs currently logged in

15) Miscellaneous Tools

archive	packs several segments together to save physical storage breakage
archive_sort	sorts the contents of an archive segment by component segment name
calc	a desk calculator
code	enciphers a segment, given a coding key
decam	another desk calculator
decode	deciphers a segment, given the proper coding key
print_motd	prints the system's message of the day
reorder_archive	rearranges order of an archive segment
who	prints list of users currently logged in

© Copyright, 1973, Massachusetts Institute of Technology and Honeywell Information Systems Inc. (END)*

TYPING CONVENTIONS

Three categories of typing conventions are dealt with in this section: canonical form, erase and kill characters, and escape characters. Quits, which might reasonably be considered a typing convention, are discussed elsewhere. The button that can be pressed to cause a quit condition to be signalled is listed for each terminal type under Terminal Switch Settings in the MPM Reference Guide section, Protocol for Logging In. Handling of the condition is discussed under the quit condition in the MPM Reference Guide section, List of System Conditions and Default Handlers.

Canonical Form

The concept of a canonical representation of a printed line image described here has been used in at least two character oriented systems: in TYPSET on the IBM 7094 (as suggested by Earl Van Horn), and in the TITAN operating system on the ATLAS computer.

Characters are intended ultimately for human communication, and conventions about a printed line must be made with this in mind. A character stream is a representation of printed lines. In general, there are many possible character streams that represent the same line. In particular, on input, a typist can produce the same printed line twice with different sets of key strokes. For example, the line

```
start    lda  alpha,4     get first result.
```

could have been typed with either spaces or horizontal tabs separating the fields; one cannot tell by looking at the printed image. Since it is not possible for the individual to distinguish between several ways of typing a printed representation, no program should deliberately attempt to do so either.

For example, a program should be able to compare easily two character streams to see if they are the same, in the sense that they produce the same printed image. It follows that all character input to Multics must be converted into a standard (canonical) form. Similarly, all programs producing character output, including editors, must produce the canonical form of output stream.

Effectively, of all possible ASCII character strings, only certain of those strings are ever found within Multics. All of those strings that produce the equivalent printed effect on a

© Copyright, 1973, Massachusetts Institute of Technology
and Honeywell Information Systems Inc.

typewriter terminal are represented within Multics as one string, the canonical form for the printed image.

No restriction has been placed on the individual at his terminal; he is free to type a noncanonical character stream. This stream is automatically converted to the canonical form before it reaches his program. For the user who wants his program to receive raw or partially processed input from his terminal, the following escape mechanism is provided. The tw_ outer module (see the MPM write-up for the tw_ subroutine) supports the following applicable modes below through the changemode call:

¬can — no canonicalization of overstrikes.

¬esc — no canonicalization of escape characters.

¬erkl — no erase and kill processing.

rawi — the specified data is read from the terminal without any conversion or processing. This includes shift characters and undifferentiated upper and lower case characters.

Similarly, an I/O System Interface Module (IOSIM) is free to rework a canonical stream on output into a different form if, for example, the different form happens to print more rapidly or reliably on the device.

We assume that every IOSIM is able to determine unambiguously what precise physical motion of the device corresponds to the actual character stream coming from or going to it. In particular, the IOSIM must know the location of physical tab settings. This requirement places a constraint on devices with movable tab stops. When the tab stops are moved, the IOSIM must be informed of the new settings. Standard Multics software assumes that tab stops are at character positions 11, 21, 31, 41, etc.

The current Multics canonical form does not meet all of the objectives of the above discussion: it represents a compromise between whose objectives and that of convenience of typing aligned tabular information, which requires an ambiguous interpretation of the tab character. The following three statements describe the current Multics canonical form.

1) A message consists of strings of character positions separated by carriage motion.

© Copyright, 1973, Massachusetts Institute of Technology and Honeywell Information Systems Inc.

2) Carriage motion consists of new line, tab, or space characters.

3) Character positions consist of a single graphic or an overstruck graphic. A character position with overstrikes contains the numerically (in ASCII sequence) smallest graphic, a backspace character, the next larger graphic, etc.

Thus, for the most part, the canonical stream differs little from the raw input stream it was derived from.

Examples of Canonical Form

Several illustrations of canonical form are shown below. The examples do not attempt to cover every conceivable variation or combination of characters, but, rather, illustrate the intent and the method. (In the examples, assume that the typist's terminal has horizontal tab stops set at 11, 21, 31, etc.)

```
Typist:          This is ordinary text.<NL>
Printed line:    This is ordinary text.
Canonical form: This is ordinary text. <NL>
```

For the case of simple, straight line input, the canonical form reduces to the original key strokes of the typist. Most input probably falls into this category.

```
Typist:          Here full<BS><BS><BS><BS>____ means that<NL>
Printed line:    Here full means that
Canonical form: Here _<BS>f_<BS>u_<BS>l_<BS>l means that<NL>
```

This is probably the most common example of canonical conversion, to insure that overstruck graphics are stored in a standard pattern.

```
Typist:          We see no prob <BS>lem<CR>__<NL>
Printed line:    We see no problem
Canonical form: _<bs>W_<bs>e see no problem
```

(Recall that the carriage return (CR) does not produce a line feed.) The most important property of the canonical form is that meanderings of the typist within a line are irrelevant. The typist need merely concern himself with the printed image.

© Copyright, 1973, Massachusetts Institute of Technology
and Honeywell Information Systems Inc.

Erase and Kill Characters

Experience has shown that even with sophisticated editor programs available, two minimal editing conventions at the earliest possible level are very useful for human input to a computer system. These two conventions give the typist two editing capabilities at the instant he is typing:

1) Ability to delete the last character or characters typed.

2) Ability to delete all of the current line typed up to this point.

(More complex editing capabilities are also available, but they fall in the domain of editing programs that can work with lines previously typed as well as the current input stream.) By framing these two editing conventions in the language of the canonical form, it is possible to preserve the ability to interpret unambiguously a typed line image despite the fact that editing was required.

The first editing convention is that one graphic, the number sign (#), is reserved as the erase character. When this character appears as the only graphic in a print position, it erases itself and the contents of the previous print position. When it is not the only graphic in a print position, it erases the print position in which it appears. If the erase character follows simple carriage motion, the carriage motion is erased. Several successive erase characters erase an equal number of print positions or simple carriage motions. Since erase processing occurs after the transformation to canonical form, there is no ambiguity as to which print position has been erased; the printed line image is always the guide. Whenever a print position is erased, any carriage motion on the two sides of the erased print position is combined into a single carriage motion.

The second editing convention reserves another graphic, the commercial at sign (@), as the kill character. When this character appears as the only graphic in a print position, the contents of that line up to and including the kill character are discarded. Again, since the kill processing occurs after the conversion to canonical form, there can be no ambiguity about which characters have been discarded. By convention, an overstruck erase character is processed before a kill character, and a kill character is processed before a non-overstruck erase character. Therefore, a kill character can only be erased by an overstruck erase character.

© Copyright, 1973, Massachusetts Institute of Technology
and Honeywell Information Systems Inc.

Examples of Erase and Kill Processing

```
Typist:          abcx#de<SP><BS>fzz##g<NL>
Printed line:    abcx#defzz##g
Canonical form: abcx#defzz##g<NL>
Final input:     abcdefg<NL>
```

This represents the primary use of the erase character, correcting typing errors the moment they are noted. Note that the erroneous space between e and f was not erased, it was undone.

```
Typist:          This@The off<BS><BS><BS>___##n<BS>_<SP>state<NL>
Printed line:    This@The off##n state
Canonical form: This@The _<BS>o_<BS>f_<BS>f##_<BS>n state<NL>
Final input:     The _<BS>o_<BS>n state<NL>
Printed appearance of final form:  The on state
```

Escape Characters

Contemporary terminal equipment often is not capable of representing all 128 of the ASCII code values. To keep full generality and flexibility in the future, standard software escape conventions are used for all terminal devices. On devices that have the revised ASCII set, the use of the escape mechanism is normally unnecessary. Each class of terminal device has a particular character assigned as the software escape character. When this character occurs in an input (or output) string to (or from) a terminal it always gives a special interpretation to the next one or more characters. The standard escape character is the left slant; this means that to input the code for it, an escape convention has to be used. Therefore, the left slant should be avoided in all Multics software. (It should be noted that the two standard erase characters, # and @, should also be avoided in all software.)

For simplicity, universal escape conventions have been established that are uniform over several terminal classes. For full flexibility, there is a mechanism for representing any arbitrary octal code in a character string. The universal escape conventions are:

\d1d2d3 for the octal code d1 d2 d3 where d1, d2, d3 are from zero to seven. \d2d3 is equivalent to \d1d2d3 if d1 is zero. \d3 is equivalent to \d1d2d3 if d1 and d2 are zero.

© Copyright, 1973, Massachusetts Institute of Technology
and Honeywell Information Systems Inc.

\<NL> for a local (i.e., concealed) use of the new line character that does not go into the computer-stored string on input and that is not in the computer-stored string on output.

\# for placing an erase character into the input string.

\@ for placing a kill character into the input string.

\\ for placing a left slant character into the input string.

One additional stylistic convention holds for all terminals: the solid vertical bar (|) and the broken vertical bar (¦) are equivalent representations of the graphic for ASCII code 174.

IBM 1050 and 2741 Terminals

Each type ball used requires a different set of escape conventions.

EBCDIC Type Ball (963) and EBCDIC Keypunches

The following non-ASCII EBCDIC graphics are considered to be stylized versions of ASCII characters:

¢	(cent sign)	for	\ (left slant, software escape)
'	(apostrophe)	for	´ (acute accent)
¬	(negation)	for	^ (circumflex)

In addition to the four universal escape conventions, the following are available for convenience.

¢'	for	`	(grave accent)
¢<	for	[	(left bracket)
¢>	for	]	(right bracket)
¢(	for	{	(left brace)
¢)	for	}	(right brace)
¢t	for	~	(tilde)

There are no currently implemented escapes for IBM 029 keypunches with EBCDIC codes.

In the case of keypunches, an end-of-card automatically generates a new line character. It is also convenient for input to have:

¢* for "skip reading the remainder of this card without the new line character"

¢/ for "new line and skip reading the remainder of this

© Copyright, 1973, Massachusetts Institute of Technology and Honeywell Information Systems Inc.

card"
¢+ for "new line and keep reading this card"
¢H for horizontal tab

Note that one can use the multiple punch codes described in the MPM Reference Guide section, Punched Card Codes, when at an 029 keypunch instead of the above escape conventions. The two sets of conventions (escape and multiple punch) are interchangeable, even in the same card deck.

Model 33 and 35 Teletypes

Because these models do not have both upper and lower case, the following typing conventions are necessary to enable users to input the full ASCII character set:

1) The keys for letters A through Z input lower case letters a through z, unless preceded by the escape character \ (left slant). The left slant is shift-L on the keyboard, although it does not show on all keyboards. For example to input "Smith.ABC", type "\Smith.\A\B\C".

2) Numbers and punctuation map into themselves when possible. The underscore (_) is represented by the back arrow (←). The circumflex (^) is represented by the up arrow (↑). The acute accent (´) is represented by the apostrophe (').

3) The following other correspondences exist:

Character	type in	octal
backspace	\-	010
grave accent (`)	\'	140
left brace ({)	\(	173
vertical line (¦)	\!	174
right brace (})	\)	175
tilde (~)	\=	176

In normal I/O mode, characters are output according to these same rules. In edited I/O mode, the output drops the left slant and becomes much tidier, but ambiguous.

Model 37 and 38 Teletypes

There are no additional escape conventions for the model 37 and 38 teletypes since they use the full ASCII character set.

© Copyright, 1973, Massachusetts Institute of Technology
and Honeywell Information Systems Inc.

Execuport 300

The following non-ASCII characters on the keyboard are considered to be stylized versions of ASCII characters:

← (back arrow)	for	_	(underscore)
↑ (up arrow)	for	^	(circumflex)

© Copyright, 1973, Massachusetts Institute of Technology and Honeywell Information Systems Inc. (END)*

COMMAND AND ACTIVE FUNCTION NAME ABBREVIATIONS

The following is a list of abbreviations for commands and active functions. The abbreviations are also listed immediately after the command or active function name in the individual write-ups. For example,

Name: abbrev, ab

Abbreviation	Command Name
aa	alm_abs
ab	abbrev
abc	adjust_bit_count
ac	archive
an	addname
as	archive_sort
bd	bind
br	basic_run
bs	basic_system
car	cancel_abs_request
cd	createdir
cdr	cancel_daemon_request
cdwd	change_default_wdir
cem	change_error_mode
cis	check_info_segs
co	console_output
cp	copy
cpa	compare_ascii
cr	create
cwd	change_wdir
da	deleteacl
db	debug
dc	deletecacl
dcm	decam
dcn	display_component_name
dd	delete_dir
df	deleteforce
did	delete_iacl_dir
dirs	directories
dis	delete_iacl_seg
dl	delete
dn	deletename
dp	dprint
dpn	dpunch
ds	dump_segment
e	enter
ear	enter_abs_request
ec	exec_com

© Copyright, 1973, Massachusetts Institute of Technology
and Honeywell Information Systems Inc.

ep	enterp
fa	fortran_abs
fl	format_line
fo	file_output
ft	fortran
gcl	get_com_line
gpn	get_pathname
gq	getquota
hd	hold
hmu	how_many_users
in	initiate
ind	indent
l	login
la	listacl
lar	list_abs_requests
lc	listcacl
lcp	lisp_compiler
ldr	list_daemon_requests
lid	list_iacl_dir
lis	list_iacl_seg
lk	link
ll	line_length
ln	listnames
lrn	list_ref_names
ls	list
lt	listotals
ml	mail
mpt	make_peruse_text
mq	movequota
mv	move
nondirs	nondirectories
nonsegs	nonsegments
pa	pl1_abs
pat	print_attach_table
pbm	print_bind_map
pdl	print_dartmouth_library
pdwd	print_default_wdir
pg	progress
pgt	page_trace
pi	program_interrupt
pli	print_link_info
plu	print_linkage_usage
pmotd	print_motd
pr	print
psr	print_search_rules
pt	peruse_text
pwd	print_wdir
qx	qedx

© Copyright, 1973, Massachusetts Institute of Technology
and Honeywell Information Systems Inc.

ra	reorder_archive
rq	reorder_archive
rdf	ready_off
rfa	runoff_abs
rf	runoff
rdn	ready_on
rdy	ready
re	reprint_error
rl	release
rn	rename
ru	resource_usage
sa	setacl
sbc	set_bit_count
sc	setcacl
scl	set_com_line
sdl	set_dartmouth_library
segs	segments
sf	sort_file
sid	set_iacl_dir
sis	set_iacl_seg
sr	start
ssd	set_search_directories
ssf	set_safety_sw_off
ssn	set_safety_sw_on
ssr	set_search_rules
st	status
tc	truncate
tm	terminate
tmr	terminate_refname
tms	terminate_segno
ts	trace_stack
ul	unlink
v2pa	v2pl1_abs
wh	where
ws	walk_subtree

© Copyright, 1973, Massachusetts Institute of Technology
and Honeywell Information Systems Inc. (END)

ALPHABETICAL LIST OF ACTIVE FUNCTIONS

The active functions of interest to most Multics users are documented in this and seven other MPM Reference Guide Sections: Logical Active Functions, Arithmetic Active Functions, Character String Active Functions, Segment Name Active Functions, Date and Time Active Functions, Question Asking Active Functions, and User Parameter Active Functions. The MPM Reference Guide section, The Command Language, describes the purpose of active functions and illustrates their use.

The following alphabetical list of the active functions available to Multics users includes the grouping to which each active function belongs; i.e., indicates which write-up contains its description.

Active Function	Group
and	logical
date	date and time
date_time	date and time
day	date and time
day_name	date and time
directories	segment name
directory	segment name
divide	arithmetic
entry	segment name
equal	logical
exists	logical
files	segment name
format_line	character string
get_pathname	segment name
greater	logical
home_dir	segment name
hour	date and time
index	character string
index_set	character string
length	character string
less	logical
links	segment name
long_date	date and time
minus	arithmetic
minute	date and time
mod	arithmetic
month	date and time
month_name	date and time
nondirectories	segment name
nonlinks	segment name
nonsegments	segment name

© Copyright, 1973, Massachusetts Institute of Technology
and Honeywell Information Systems Inc.

not	logical
or	logical
path	segment name
pd	segment name
plus	arithmetic
query	question asking
response	question asking
segments	segment name
string	character string
strip	segment name
strip_entry	segment name
substr	character string
suffix	segment name
time	date and time
times	arithmetic
unique	segment name
user	user parameter
wd	segment name
year	date and time

© Copyright, 1973, Massachusetts Institute of Technology
and Honeywell Information Systems Inc. (END)

USER PARAMETER ACTIVE FUNCTIONS

The active functions of interest to most Multics users are documented in this and seven other MPM Reference Guide sections: Alphabetical List of Active Functions, Logical Active Functions, Arithmetic Active Functions, Character String Active Functions, Segment Name Active Functions, Date and Time Active Functions, and Question Asking Active Functions. The MPM Reference Guide section, The Command Language, describes the purpose of active functions and illustrates their use. Note that in a command line an active function must be enclosed in square brackets. However, those brackets have been omitted from usage descriptions in this write-up. They are included in examples.

The following active function returns user parameters obtained from system data bases.

Name: user

This active function returns various user parameters.

Usage

user arg

1) arg — can have one of the following values:

name	returns the user name at log in time.
project	returns the user project ID.
login_date	returns the date at log in time. The date is of the form "mm/dd/yy".
login_time	returns the time of log in. The time is of the form "hhmm.t".
anonymous	returns "true" if the user is an anonymous user; otherwise it returns "false".
secondary	returns "true" if the user is currently subject to preemption; otherwise it returns "false".
absentee	returns "true" if the user is an absentee user; otherwise it returns "false".
term_id	returns the user's terminal ID code. It is "none" if the user's terminal does not have

© Copyright, 1973, Massachusetts Institute of Technology and Honeywell Information Systems Inc.

the answer back feature.

term_type — returns the user's terminal type. It can have one of the following values:

"Absentee"
"Network"
"1050"
"2741"
"IBM2741"
"TTY33"
"TTY37"
"TN300"
"ARDS"

The terminal types "2741" and "IBM2741" differ in that "IBM2741" designates a standard IBM 2741 terminal, and "2741" designates a 2741 terminal that has been modified according to MIT specifications. The modification prevents keyboard locking after a carriage return. A "2741" terminal can be made to look exactly like an "IBM2741" terminal by placing its INHIBIT AUTO EOT switch in the off position.

cpu_secs — returns the user's CPU usage, in seconds, since log in. The usage is of the form "sss.t" with leading zeros suppressed.

log_time — returns the user connect time, in minutes, since log in. The time is of the form "mmm.t".

preemption_time — if the user is a primary user, returns the time at which he becomes eligible for group preemption. The time is of the form "hhmm.t".

brief_bit — returns "true" if the user specified the -brief control argument in his login line; otherwise, returns "false".

protected — if the user is currently a primary user and protected from preemption, returns "true"; otherwise, returns "false".

© Copyright, 1973, Massachusetts Institute of Technology
and Honeywell Information Systems Inc.

absin — if the user is an absentee user, this returns the absolute path name of his absentee input segment including the .absin suffix; otherwise returns a null string.

absout — if the user is an absentee user, returns the absolute path name of his absentee output segment; otherwise, returns a null string.

Example

The following example illustrates the use of one of the active functions described in this write-up.

```
ioa_ [user login_time]
```

This example causes the time the user logged in to be printed at the user's terminal.

© Copyright, 1973, Massachusetts Institute of Technology
and Honeywell Information Systems Inc. (END)*

GRAPHICS SUPPORT ON MULTICS

The basic purpose behind the development of the Multics Graphic System is to provide a terminal-independent general purpose graphic interface suitable for use by all graphic applications on Multics. A graphic program written at one type of graphic terminal should be operable at another type of terminal of similar capabilities without modification. A wide variety of graphic applications should be supported. The user should be able to write his program easily and naturally, and it should run in an efficient manner.

This first attempt at a graphic system for Multics naturally represents compromises with the above goals. The Multics Graphic System will change as better solutions to the problems involved are discovered. These changes will be upward compatible whenever possible, and every attempt will be made to support graphic programs written for previous versions of the graphic system.

The idea of terminal independence was adopted from the Multics typewriter I/O system, for the same reason that it is so important there. A wide variety of console types are connected to Multics (including graphic terminals used as video typewriters), and this console mix changes with time. To require a program to incorporate specific terminal dependencies would have severe disadvantages.

a) User Fragmentation. Only a limited amount of user program sharing could take place because a program written by a user could not be used by others working at different terminals. This kind of fragmentation would severely inhibit the development of on-line user communities.

b) Terminal Immobility. Being able to use only a fraction of the terminals connected to the utility is a considerable inconvenience. Even worse, the dependence of a subsystem on a specific terminal would grossly inhibit the transference to new and better types of terminals.

c) Certain modules provided by the utility would require a different version for each supported terminal type.

The net result would be to support only a limited number of different types of terminals, to inhibit the introduction of new types, and to retain an obsolete type long after the desirability of its removal had become apparent.

© Copyright, 1971, Massachusetts Institute of Technology
All rights reserved.

While these arguments for terminal independence now apply most strongly to typewriters, as more types of graphic terminals are added to Multics, they gain validity for graphics as well.

What is meant by terminal independence in graphics? There are many different types of graphic terminals ranging from storage tube terminals to refresh displays with internal processing capability. Even a plotter could be connected, though not as an interactive console. Each device has its own set of special features, such as dotted line or blinking capability. To create true device indifference by allowing the use only of those features common to all terminals is clearly an intolerable solution. Rather, as a compromise solution, an interface incorporating the union of all features of existing terminals is provided. This interface is extensible to include new features of terminals to be attached in the future.

A user tailors his program to use the features of the terminal types he intends to use. When the program is run, the use of any unavailable feature is mapped into the most reasonable compromise feature of the terminal being operated. Thus, the user has a reasonable guarantee that a program coded in accord with the limitations of the system's terminal independence capability will generate a recognizable picture on practically any type of graphic terminal attached to Multics. However, it will not necessarily operate equally well at any terminal. For example, a program written to use the dynamic rotation and editing capabilities of a programmable display such as an IMLAC would operate rather poorly on a storage tube display terminal such as the ARDS.

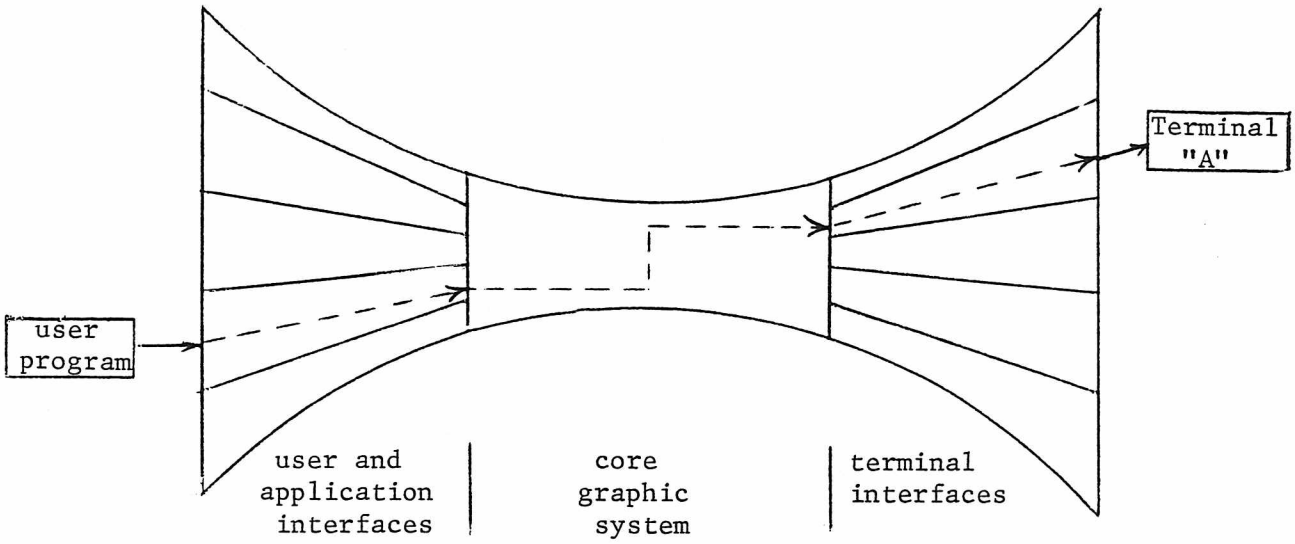
The motivation to provide a general purpose graphic system is to avoid creating and maintaining a multiplicity of systems, each oriented towards a separate application. The extra overhead borne by the utility is obvious, but also important is the added burden of graphic users having to master the idiosyncrasies of entirely separate systems.

The design problem of any general purpose system is that generality is often opposite to ease and efficiency of use. A system intended to accept a wide variety of tasks may perform few of them well. The Multics Graphic System avoids this dilemma in a compromise fashion. It provides a sophisticated, picture-structure oriented programming interface which is suitable for direct use by a knowledgeable programmer. However, those desiring a more application oriented or simpler interface may use instead application modules that sit between a user program and the general graphic interface. The programming

© Copyright, 1971, Massachusetts Institute of Technology
All rights reserved.

interfaces of these application modules present a view of the graphic system more structured towards the user's needs. Currently, both a casual user module and a plotting module exist.

Conceptually, the Multics Graphic System can be viewed as a double-ended funnel implementing a switching function between a particular application interface and a particular terminal type.



A Conceptual View of the Organization
of the Multics Graphic System

© Copyright, 1971, Massachusetts Institute of Technology
All rights reserved.

The central portion is the core graphic system, through which everything is channeled. At the user end, the funnel expands to include a number of different application interface modules. The user is to select the most appropriate of these for use by his graphic program. At the system end, the funnel expands to include interface modules for all the different graphic terminals attached to the utility. The user exchanges graphic communications with his terminal through one of these.

More extensive graphic system information and graphic system module write-ups are available in the MPM Graphics Users' Supplement.

© Copyright, 1971, Massachusetts Institute of Technology
All rights reserved.

(END)*

Control Characters

The following conventions define the standard meaning of the ASCII control characters which are given precise interpretations in Multics. These conventions will be followed by all standard Device Interface Modules (DIM) and by all system software inside the I/O system interface. Since some devices have different interpretations for some characters, it is the responsibility of the appropriate DIM to perform the necessary translations.

The characters designated as "not used" are specifically reserved and may be assigned definitions at any time. Until defined, "not used" control characters will be output using the octal escape convention in normal output and not printed in edited mode. Users wishing to assign interpretations for a "not used" character must use a non-standard DIM.

If a device does not perform a function implied by a control character, its standard DIM will provide a reasonable interpretation for the character on output. This may be substituting one or more characters for the one, or printing an octal escape, or ignoring it.

The Multics standard control characters are:

BEL Sound an audible alarm.

BS Backspace. Move the carriage back one space. The backspace character implies overstrike rather than erase.

HT Horizontal tab. Move the carriage to the next horizontal tab stop. Multics standard tab stops are at 11, 21, 31... when the first column is numbered 1. This character is defined not to appear in a canonical string.

NL "New line". Move the carriage to the left end of the next line. This implies a carriage return plus a line feed. ASCII LF (octal 012) is used for this character.

VT Vertical tab. Move the carriage to the next vertical tab stop and to the left of the page. Standard tab stops are at lines 11, 21, 31... when the first line is numbered 1. This character is defined not to appear in a canonical string.

NP New page. Move the carriage to the top of the next page and to the left of the line. ASCII FF (octal 014) is used for this character.

© Copyright, 1971, Massachusetts Institute of Technology
All rights reserved.

ASCII CHARACTER SET

The Multics standard character set is the revised U.S. ASCII Standard (refer to USA Standards Institute, "USA Standard X3.4-1968"). The ASCII set consists of 128 seven bit characters. Internally these are stored right justified in four nine bit fields per word. The two high order bits in each field are expressly reserved for expansion of the character set; no system program shall use them. Any hardware device which is unable to accept or create the full character set should use established escape conventions for representing the entire set. It is emphasized that there are no meaningful subsets of the revised ASCII character set.

Included in the ASCII character set are 94 printing graphics, 33 control characters, and the space. Multics conventions assign precise interpretations to all the graphics, the space, and 11 of the control characters. One of these control characters is the "Enter Graphic Mode" character which is recognized only on a graphics device. The remaining 22 control characters are presently reserved. The graphics in the set are as follows:

Upper Case Alphabet

ABCDEFGHIJKLMNOPQRSTUVWXYZ

Lower Case Alphabet

abcdefghijklmnopqrstuvwxyz

Digits

0123456789

Special Characters

! exclamation point
" double quote
# number sign
$ dollar sign
% percent
& ampersand
' acute accent
(left parenthesis
) right parenthesis
* asterisk
+ plus
, comma
- minus
. period
/ right slant
: colon
; semicolon
< less than
= equals
> greater than
? question mark
@ commercial at
[left bracket
\ left slant
] right bracket
^ circumflex
_ underline
` grave accent
{ left brace
| vertical line
} right brace
~ tilde

© Copyright, 1971, Massachusetts Institute of Technology
All rights reserved.

CR Carriage return. Move the carriage to the left of the current line. This character is defined not to appear in a canonical string.

RRS Red ribbon shift. ASCII SO (octal 016) is used for this character.

BRS Black ribbon shift. ASCII SI (octal 017) is used for this character.

PAD Padding character. This is used to fill out words which contain fewer than four characters and which are not accompanied by character counts. This character is discarded when encountered in an output line and cannot appear in a canonical character string. ASCII DEL (octal 177) is used for this character.

Non-Standard Control Character

One control character is recognized under certain conditions by all DIMs because of its wide use outside Multics. This character is handled specially only when the DIM is printing in edited mode, i.e., it is ignoring unavailable control functions. This character is

NUL Null character. ASCII character NUL (octal 000) is used for this purpose. In normal mode, this character is printed with an octal escape sequence; in edited mode, it is treated exactly as PAD. This character cannot appear in a canonical character string.

Programmers are warned against using NUL as a routine padding character and using edited mode on output because all strings of zeros, including mistakenly uninitialized strings, will be discarded.

Graphic Control Characters

Character code 037 octal (ASCII US) is used to escape into graphic output mode when a graphic display terminal is in use; it is treated as "not used" and an octal escape is provided when an ordinary typewriter is in use. Details on Multics graphic output mode may be found in the Graphic Users' Supplement to the MPM.

© Copyright, 1971, Massachusetts Institute of Technology
All rights reserved.

ASCII Character Set
Standard Data Formats and Codes
Page 4

Not Used Characters

These characters are reserved for future use:

SOH	001	ACK	006	DC4	024	EM	031
STX	002	DLE	020	NAK	025	SUB	032
ETX	003	DC1	021	SYN	026	ESC	033
EOT	004	DC2	022	ETB	027	FS	034
ENQ	005	DC3	023	CAN	030	GS	035
						RS	036

Notes

The vertical line has two representations on current console devices. It may be represented as either a solid vertical line (|) or a broken vertical line (¦). These are represented identically internally by octal value 174.

© Copyright, 1971, Massachusetts Institute of Technology
All rights reserved.

ASCII Character Set on Multics

	0	1	2	3	4	5	6	7
000	(NUL)							BEL
010	BS	HT	NL	VT	NP	CR	RRS	BRS
020								
030								EGM
040	Space	!	"	#	$	%	&	'
050	(	)	*	+	,	-	.	/
060	0	1	2	3	4	5	6	7
070	8	9	:	;	<	=	>	?
100	@	A	B	C	D	E	F	G
110	H	I	J	K	L	M	N	O
120	P	Q	R	S	T	U	V	W
130	X	Y	Z	[	\	]	^	_
140	`	a	b	c	d	e	f	g
150	h	i	j	k	l	m	n	o
160	p	q	r	s	t	u	v	w
170	x	y	z	{	¦	}	~	PAD

© Copyright, 1971, Massachusetts Institute of Technology
All rights reserved.

ASCII Character Set
Standard Data Formats and Codes
Page 6

Multics Definitions

NUL	Null character (edited output mode only)
BEL	Alarm
BS	Backspace
HT	Horizontal Tab
NL	New Line (carriage return and line feed)
VT	Vertical Tab
NP	New Page (carriage return and form feed)
CR	Carriage Return
RRS	Red Ribbon Shift
BRS	Black Ribbon Shift
EGM	Enter Graphic Mode
PAD	Padding Character

© Copyright, 1971, Massachusetts Institute of Technology
All rights reserved.

(END)*

THE LIMITED SERVICE SYSTEM

A small set of Multics commands, all of which are noted for their light resource usage, compose what is known as the Limited Service System (LSS). A user of the LSS is prevented by the system from using commands outside this set; in addition, his rate of CPU usage is tightly controlled.

The LSS provides a way for a project to allow its members (or a subset of its members) access to computing services with a ceiling on the amount of money they will spend. For example, a professor could leave a terminal logged in all day, available to anyone in his class, and still be sure that only a fixed amount of money would be spent each day.

To register an LSS user, the project administrator need only specify the LSS process overseer as that user's process overseer.

The LSS is sufficiently modular that pieces of it (for instance, the piece that controls which commands are accessible) may be used in other subsystems. This means that subsystem writers can easily tailor their own limited systems with their own list of commands and/or governing parameters. Documentation of the subroutines necessary to do this are published in the MPM Subsystem Writers' Supplement (SWS).

The following commands are currently allowed in the LSS:

addname
basic
basic_system
calc
decam
delete
deletename
edm
help
list
listnames
listotals
logout
print
program_interrupt
ready
ready_off
ready_on
rename
start

© Copyright, 1972, Massachusetts Institute of Technology
All rights reserved.

(END)

CB1 61 5x2

LIST OF NAMES WITH SPECIAL MEANINGS

The following names are reserved for special purposes within Multics. The user should not use them with a different meaning. See also the following MPM Reference Guide sections for other names with special meanings: List of System Conditions and Default Handlers, List of Names in the System Libraries, and Obsolete Procedures.

Reserved I/O Stream Names

By convention, the following I/O stream names are reserved. Those maintained by the standard environment are:

user_i/o	is the stream attached to the user's terminal or absentee input and output segments.
user_input	is the stream attached to user_i/o and devoted expressly to read calls.
user_output	is the stream attached to user_i/o and devoted expressly to write calls.
error_output	is the stream attached to user_i/o and devoted expressly to write calls under error conditions.

Those maintained by system commands or subroutines are:

exec_com_stream_N	is the stream attached by the exec_com command using the attach command line. N is a unique sequence number assigned by exec_com. user_input is attached to this stream through the syn interface module.
file_output_stream_	is the stream attached by the file_output command. user_output is attached to this stream through the syn interface module.
graphic_input	is the stream used for graphics input.
graphic_output	is the stream used for graphics output.

© Copyright, 1973, Massachusetts Institute of Technology
and Honeywell Information Systems Inc.

Reserved Segment Names

By convention, the following segment names are reserved. Those maintained in the home directory are:

mailbox	is the segment used by the mail command.
start_up.ec	is the exec_com invoked at the beginning of a process in the standard environment.
username.breaks	is the break segment used by the debug command. (username is the name derived from the login command.)
username.con_msgs	is the segment used by the message facility (see the MPM write-up of the send_message command). (username is the name derived from the login command.)
username.memo	is the segment used by the memo command. (username is the name derived from the login command.)
username.motd	is the segment used by the print_motd command. (username is the name derived from the print_motd command.)
username.profile	is the segment used by the abbrev command. (username is the name derived from the login command.)

Those maintained in the process directory are:

combined_linkage_N.jk	is the user's linkage segment for ring number N (1<=N<=7). jk is a two digit sequence number. This segment also contains internal static storage.
drum_temp1_ drum_temp2_	is temporary storage used by alm, edm, and qedx.
kst	(Known Segment Table) is a Hardcore Ring data segment.
pds	(Process Data Segment) is a Hardcore Ring data segment.

© Copyright, 1973, Massachusetts Institute of Technology and Honeywell Information Systems Inc.

pit	is the user's Process Initialization Table. It should only be referenced through the subroutine user_info_ (see the MPM subroutine write-up for user_info_).
stack_N	is the user's automatic storage area for ring number N (1<=N<=7).
system_free_N_	is the free storage area used by system commands for ring number N (1<=N<=7).

In general, users should not create segments whose names end in a trailing underscore (_). These names are reserved for system subroutines and may cause errors if they are in the user's search path. (See the MPM Reference Guide section, The System Libraries and Search Rules.)

Reserved Segment Name Suffixes

Suffixes are used as in the following example: If one is creating a PL/I source program to be named xyz, he would create a source language segment named xyz.pl1. The PL/I compiler, by convention, translates this segment, producing the segment xyz.list, containing a printable listing, and the segment xyz, containing the object program.

By convention, the following segment name suffixes are reserved. The language translator source segment suffixes are:

Language Translator	Source Segment	Include Files
PL/I compiler	.pl1	.incl.pl1
FORTRAN compiler	.fortran	.incl.fortran
ALM assembler	.alm	.incl.alm
BASIC compiler	.basic	

The listing segment suffix is:

.list	is the suffix on printed output listing segments produced by compilers, the assembler, and the binder.

© Copyright, 1973, Massachusetts Institute of Technology and Honeywell Information Systems Inc.

Other special suffixes are:

.absin	is the input segment suffix for an absentee process.
.absout	is the default output segment suffix for an absentee process.
.apl	is the suffix on the segment containing a saved workspace from the apl command.
.archive	is the suffix on the segment created by the archive command.
.bind	is the suffix on the input control segment for the binder.
.dobj	is the suffix on the output segment produced from the -compile control argument to the BASIC compiler.
.ec	is the suffix on the input segment to the exec_com command.
.info	is the suffix on a segment, in >documentation>info_segments, for use with the help command.
.lisp	is the suffix on the segment containing a saved environment from the lisp command.
.ms	is the suffix on an Administrative Ring message segment.
.pt	is the segment suffix for use with the peruse_text command.
.runoff	is the input segment suffix to the runoff command.
.runout	is the output segment suffix from the runoff command.

Reserved Object Segment Entry Point

By convention, the following entry point definition in object segments is reserved.

© Copyright, 1973, Massachusetts Institute of Technology
and Honeywell Information Systems Inc.

symbol_table	is the entry point definition which provides the address of the symbol table produced by the pl1 or fortran commands.

Since this is a reserved entry point, no user-created program can use this name. A statement of the form

```
symbol_table: procedure...
```

is illegal if it is the external procedure block.

© Copyright, 1973, Massachusetts Institute of Technology
and Honeywell Information Systems Inc. (END)*

Command
Development System
5/17/72

Name: answer

This command provides a preset answer to any questions asked by some other command. It does this by establishing a handler for the condition command_question, and then executing the subject command. If the subject command calls command_query_ to ask a question, the handler will be invoked to supply the answer. The handler is reverted when "answer" exits.

Usage

answer ans -control_args- commandline

1) ans — is the desired answer to any question.

2) control_args — may be chosen from the following list of control arguments:

-brief, -bf — suppresses printing of both the question and the answer.

-times n — gives the prespecified answer n times only (where n is an integer); then it acts as if "answer" had not been called.

3) commandline — is any Multics command line.

Notes

If a question is asked which requires a yes or no answer, and the preset answer is neither yes nor no, the handler will not be invoked.

Examples

To delete a directory without being asked if you want to delete it:

```
answer yes -bf dd test_dir
```

To see the first three blocks of an info segment named fred.info, and then be interrogated:

```
answer yes -times 2 help fred
```

To see only the first three blocks:

© Copyright, 1972, Massachusetts Institute of Technology
All rights reserved.

```
answer no answer yes -times 2 help fred
```

In the above example, "answer" is invoked twice. The first invocation is to answer no and is given the command line

```
answer yes -times 2 help fred
```

The second invocation is to answer yes twice, and is given the command line

```
help fred
```

The help command prints the first block of fred.info, and gets the answer yes from the second invocation of "answer". It repeats this process, and again obtains a yes answer. After help prints the third block of fred.info, however, the second invocation of "answer" has had its count run out, and behaves as if it had not been called. Hence, the first invocation of "answer" supplies the answer no and execution ends.

© Copyright, 1972, Massachusetts Institute of Technology
All rights reserved.

(END)

Command
Standard Service System
3/17/72

Name: copy, cp

The copy command causes copies of indicated segments to be created in indicated directories with indicated names. Access control lists (ACLs) and multiple names are optionally copied.

Usage

copy path1*1* path2*1* ... path1*n* path2*n* -ca*1*- ... -ca*n*-

1) path1*i* are the segments to be copied.

2) path2*i* are the copies to be created. If the last path2*i* is not given, the copy is placed in the working directory with the entry name of the segment from which it was copied.

3) ca*i* may be chosen from the following list of control arguments:

-name, -nm causes multiple names to be copied.

-acl causes the ACL to be copied.

-all, -a causes multiple names and ACLs to be copied.

-brief, -bf causes the messages "Bit count inconsistent with current length..." and "Current length is not the same as records used..." to be suppressed.

The control arguments may appear once anywhere in the command line and apply to the entire command line.

Notes

Read access is required for path1*i*. Execute access is required for the directory containing path1*i*. Execute, write, and append access are required for the directory containing path2*i*.

The star and equal conventions may be used.

Example

```
copy >old_dir>fred.list george.=
```

The segment fred.list in the directory >old_dir is copied into the working directory as george.list.

© Copyright, 1972, Massachusetts Institute of Technology
All rights reserved.

(END)

Name: do

The purpose of the do command is to expand a command string given as its first argument by replacing the parameter designators &1 ... &9 found in it with the actual arguments supplied following the command string. The resultant expanded string is then passed to the Multics command processor for execution. If abbreviations are being expanded in the process, any abbreviations in the expanded string are first expanded. (See the writeup for the abbrev command.) Control arguments exist to print the expanded command line, to suppress its execution, or to pass it back as the value of an active function.

Usage

do "command_string" -arg1- ... -argn-

1) command_string — is a command line (in quotation marks). Each instance of the paramter designator &i (where i is a number from 1 to 9) found in command_string is replaced by the corresponding actual argument argi. If any argi is not supplied, then each instance of &i in command_string is replaced by the null string. Each instance of the unique-name designator &! found in command_string is replaced by a 15-character identifier unique to the particular invocation of the do command. Finally, each instance of the pair && is replaced by a ampersand. Any other ampersand discovered in command_string causes an error message to be printed and the expansion to be terminated. Any argument supplied but not mentioned in a paramter designator is ignored.

2) argi — is a character string argument to replace a parameter designator &i in command_string.

Usage as an Active Function

If the do command is called as an active function

|do "command_string" arg1 ... argn|

then, instead of executing the resultant expanded string, the do command passes it back as the value of the active function.

© Copyright, 1973, Massachusetts Institute of Technology
and Honeywell Information Systems Inc.

Modes

The do command has three modes, the long/brief mode, the nogo/go mode, and the absentee/interactive mode. These modes are kept in internal static storage and are thus remembered from call to call within a process. The modes are set by invoking the do command as follows:

do mode

where mode is one of -long, -lg, -brief, -bf, -nogo, -go, -absentee, or -interactive.

If the long/brief mode is long, then the expanded command string is printed on the string error_output before it is executed or passed back. If the long/brief mode is brief, then the string is not printed. The default for this mode is brief.

If the nogo/go mode is nogo, then the expanded command string is not passed to the command processor for execution. If the nogo/go mode is go, then the expanded string is passed to the command processor (if the do command was invoked as a command). If do is invoked as an active function, then the nogo/go mode is ignored. The default for this mode is go.

If the absentee/interactive mode is absentee, then the do command establishes itself as a default on unit during the execution of the expanded command string. This is mainly of use in an absentee environment, in which any invocation of the standard default on unit terminates the process. When do is the default on unit, any signal caught by do merely terminates execution of the command string, not the process. A number of conditions, however, are not handled by do but are passed on for their standard Multics treatment; they are cput, alrm, quit, program_interrupt, command_error, command_query_error, command_question, and record_quota_overflow. (For a description of these conditions see the MPM Reference Guide section, List of System Conditions and Default On Unit Actions.) If the absentee/interactive mode is interactive, then do does not catch any signals. The default for this mode is interactive.

Quote-doubling and Requoting

In addition to the parameter designators &1 ... &9, the do command also recognizes two more sets of parameter designators. They are &q1 ... &q9, to request quote-doubling in the actual argument as it is substituted into the expanded string, and &r1 ... &r9, to request that the actual argument be requoted as well as have its quotes doubled during substitutution.

© Copyright, 1973, Massachusetts Institute of Technology and Honeywell Information Systems Inc.

Quote-doubling can be described as follows. Each parameter designator in the input string to be expanded is found nested a certain level deep in quotes. If a designator is found to be not within quotes, then its quote-level is zero; if it is found between a single pair of quotes, then its quote-level is one; and so on. If the parameter designator &qi is found nested to quote-level L, then, as argi is substituted into the expanded string each quote character found in argi is multiplied by 2**L during insertion. This permits the quote character to survive the quote-stripping action to which the command processor subsequently subjects the expanded string. If &qi is not located between quotes, or if argi contains no quotes, then the substitutions performed for &qi and for &i are identical.

If the parameter designator &ri is specified, then the substituted argument argi is placed between an additional level of quotes before having its quotes doubled. More precisely, if the parameter designator &ri is found nested to quote-level L, then 2**L quotes are inserted into the expanded string, argi is substituted into the expanded string with each of its quotes multiplied by 2**(L+1), and then 2**L more quotes are placed following it. If argument argi is not supplied, then nothing is placed in the expanded string; this providis a way to distinguish between arguments that are not supplied and arguments that are supplied but null. If argument argi is present, then the expansions of &ri and of &qi written between an additional level of quotes are identical.

Accessing More than Nine Arguments

In addition to the normal parameter designators in which the argument to be substituted is specified by a single digit, do also allows the designators &(d...d), &r(d...d), and &q(d...d) where d...d denotes a string of decimal digits. An error message is printed and the expansion is terminated if any character other than 0 ... 9 is found between the parentheses.

Examples

The do command is particularly useful when used in conjunction with the abbreviation processor, the abbrev command. Consider the following abbreviations:

```
ADDPLI   do "fo &1.list;ioa_ ¬|;pli &1;co"
AUTHOR   do "ioa_$nnl &1;status -author $1"
AX       do "if is &1 -then ""df &1"""
FO       do "AX &1;fo &1"
P        do "pl1 &1 -list &2 &3"
```

© Copyright, 1973, Massachusetts Institute of Technology
and Honeywell Information Systems Inc.

The command

```
ADDPLI alpha
```

expands to

```
fo alpha.list;ioa_ ^|;pli alpha;co
```

The command line

```
AUTHOR beta
```

prints the author of segment beta.

The command line

```
FO gamma
```

expands to

```
AX gamma;fo gamma
```

which is expanded to

```
if is gamma -then df gamma;fo gamma
```

This shows how do can be used at several levels and how abbreviations can be used within abbreviations.

The command line

```
P alpha
```

generates the expansion

```
pl1 alpha -list
```

while the command line

```
P alpha -table
```

expands to

```
pl1 alpha -list -table
```

This shows how references to unsupplied arguments get deleted.

© Copyright, 1973, Massachusetts Institute of Technology
and Honeywell Information Systems Inc. (END)

Name: edm

The edm command is the standard Multics context editor. It is used for creating and editing ASCII segments. This command cannot be called recursively. See also the MPM Introduction Chapter 3, Beginner's Guide to the Use of Multics.

Usage

edm -pathname-

1) pathname specifies the segment to be edited. The argument is optional. If pathname is not given, edm will begin in input mode (see Notes below), ready to accept whatever is typed subsequently as input; when the created segment is written out, its name may be specified as part of the write request. If pathname is given, but the segment does not yet exist, edm will also begin in input mode; otherwise, edm will begin in edit mode, ready to edit the segment specified by pathname.

Notes

This command operates in response to requests from the user. To issue a request, the user must cause edm to be in edit mode. This mode is entered in two ways: if the segment already exists, it is entered automatically when edm is invoked; if dealing with a new segment (and edm has been in input mode), the mode change characters must be issued. The mode change characters are the period (.) followed by a "new line" (carriage return-new line), issued as the only characters on a line. The command announces its mode by typing "Edit." or "Input." when the mode is entered. From edit mode, input mode is also entered via the mode change characters.

The edm requests are predicated on the assumption that the segment consists of a series of lines to which there is a conceptual pointer which indicates the current line. (The "top" and "bottom" lines of the segment are also meaningful.) Various requests explicitly or implicitly cause the pointer to be moved; other requests manipulate the line currently pointed to. Most requests are indicated by a single character; for these, the character is generally the first letter of the name of the request. Only the single character is accepted by the command. Three requests have been considered sufficiently dangerous, or likely to confuse the unwary user, that their names must be specified in full.

© Copyright, 1973, Massachusetts Institute of Technology
and Honeywell Information Systems Inc.

If the user presses the quit button while in edit mode and then invokes the program_interrupt command, the effect of the last request executed on the edited copy is nullified. (See the MPM write-up for program_interrupt.) In addition any requests not yet executed are lost. If program_interrupt is typed after a quit in comment or input modes then all input since last leaving edit mode will be lost. If the user wishes to keep the input he must type start following the quit.

Requests

The requests are as follows (detailed descriptions follow the list, in the order of the list):

-	backup
=	print current line number
,	comment mode
.	mode change
b	bottom
c	change
d	delete
E	execute
f	find
i	insert
k	kill
l	locate
merge	insert segment (move)
n	next
p	print
q	quit
qf	quitforce
r	retype

© Copyright, 1973, Massachusetts Institute of Technology
and Honeywell Information Systems Inc.

s	substitute
t	top
updelete	delete to pointer
upwrite	write to pointer (upper portion of segment)
v	verbose
w	write

- Request

Format: - n

Purpose: Move pointer backwards (toward the top of the segment) the number of lines specified by the integer n.

Spacing: A space is optional between the request and the integer argument.

Pointer: Set to the nth line specified before the current line.

Default: If n is null, the pointer is moved up only one line.

Example:

```
Before:     a:  procedure;
                x = y;
                q = r;
                s = t;
            ->  end a;

Request:    -2

After:      a:  procedure;
                x = y;
            ->  q = r;
                s = t;
                end a;
```

© Copyright, 1973, Massachusetts Institute of Technology
and Honeywell Information Systems Inc.

= Request

Format: =

Purpose: Print current line number.

Pointer: Unchanged.

, Request

Format: ,

Purpose: The editor will print lines, starting with the current one, leaving off the carriage return. It then switches to input mode, letting you type the rest of the line (comment, "new line", etc.). The process then repeats with the next line. The mode change characters will return you to edit mode.

Pointer: Left pointing to the last line printed.

b Request

Format: b

Purpose: Move pointer to the end of the segment and switch to input mode.

Pointer: Set after the last line in the segment.

c Request

Format: c n /string1/string2/

Purpose: Replace every instance of string1 by string2 in the number of lines indicated to be searched by the integer n. edm responds to each change by printing the line with the changed text in red if the user is in verbose mode (see the v request), or with "edm: Substitution failed." if string1 is not found.

Spacing: A space before n and between n and the string1 delimiter is optional.

Pointer: Set to the last line scanned.

© Copyright, 1973, Massachusetts Institute of Technology
and Honeywell Information Systems Inc.

Delimiters: Any character not appearing in string1 or string2 can delimit the strings (/ is shown in the format). A delimiter following string2 is optional.

Default: If an integer is absent, only string1 of the current line is changed. If string1 is absent, string2 is inserted at the beginning of the line.

Example:

```
Before:     a:  procedure;
            ->  x = y.
                q = r.
                s = t;
                end a;

Request:    c2/./;/

Response:   x = y;
            q = r;

After:      a:  procedure;
                x = y;
            ->  q = r;
                s = t;
                end a;
```

Note: For compatibility with qedx, this request may also be given as s (for substitute).

d Request

Format: d n

Purpose: Causes n lines to be deleted where n is an integer. Deletion begins at the current line.

Spacing: A space is optional between d and n.

Pointer: Set to "no line" following the line deleted. That is, an i request or a change to input mode would take effect before the next nondeleted line.

Default: If n is null, only the current line is deleted.

Note: The requests c, d, n, and p count "no line" when issued immediately after a delete request.

© Copyright, 1973, Massachusetts Institute of Technology
and Honeywell Information Systems Inc.

E Request

Format: E commandline

Purpose: Pass commandline to the command processor for execution as a command line.

Spacing: A single space following E is not significant.

Pointer: Unchanged.

f Request

Format: f string

Purpose: Search segment for a line beginning with the string. Search starts at the line following the current line and continues around the entire segment until the string is found or until return to the current line. The current line is not searched. If the string is not found, the error message "edm: Seach failed." is printed. If the string is found and the user is in verbose mode, the line containing the string is printed.

Spacing: A single space following f is not significant. All other leading and embedded spaces are used in searching.

Pointer: Set to the line found, or remains at the current line if the line is not found.

Default: If the string is null, edm searches for the string requested by the last f or l request.

i Request

Format: i newline

Purpose: Insert newline after the current line.

Spacing: The first space following i is not significant. All other leading and embedded spaces become part of the text of the new line.

Pointer: Set to the inserted line.

Default: If newline is null, a blank line is inserted.

© Copyright, 1973, Massachusetts Institute of Technology
and Honeywell Information Systems Inc.

Note: Immediately after a t (top) request, an i request causes the newline to be inserted at the beginning of the segment.

k Request

Format: k

Purpose: To inhibit (kill) responses following a c, f, l, n, or s request.

Pointer: Unchanged.

Note: See v (verbose) request for restoring responses.

l Request

Format: l string

Purpose: Search segment for a line containing the string. Search starts at the line following the current line and continues around the entire segment until the string is found or until return to the current line. The current line is not searched. If the string is not found, the error message "edm: Search failed." is printed. If the string is found and the user is in verbose mode, the line containing the string is printed.

Spacing: A single space following l is not significant. All other leading and embedded spaces are used in searching.

Pointer: Set to the line found, or remains at the current line if the line is not found.

Default: If the string is null, edm searches for the string requested by the last l or f request.

Example:

Before:
```
    a:  procedure;
        x = y;
        q = r;
->      s = t;
        end a;
```

Request: l x =

© Copyright, 1973, Massachusetts Institute of Technology
and Honeywell Information Systems Inc.

```
After:      a:  procedure;
            ->  x = y;
                q = r;
                s = t;
                end a;
```

merge Request

Format: merge path

Purpose: The segment specified by path is inserted after the current line.

Spacing: A single space following merge is not significant.

Pointer: Set to the last line of the inserted segment.

Default: If path is not given, the name given in the invocation of edm is used.

n Request

Format: n n

Purpose: Move pointer down the segment n lines.

Spacing: A space is optional between n and the integer n.

Pointer: Set to the nth line specified after the current line.

Default: If the integer n is null, the pointer is moved down only one line.

Note: The printed response to this request can be shut off using the k request.

p Request

Format: p n

Purpose: n lines will be printed, beginning with the current line.

Spacing: A space is optional between p and the integer n.

Pointer: Left pointing to the last line printed.

Default: If n is null, the current line is printed.

© Copyright, 1973, Massachusetts Institute of Technology
and Honeywell Information Systems Inc.

Note: A print request in edm may be aborted by pressing the quit button and typing pi or program_interrupt. This will put edm in a state where it is ready to accept another request. (See the MPM write-up for program_interrupt.)

q Request

Format: q

Purpose: To exit edm and return to the caller, usually command level. If no write request has been made since the last change to the edited text, edm will warn the user that the changes will be lost and ask if he still wishes to quit.

Pointer: If the user is queried and answers no, then the pointer is unchanged.

qf Request

Format: qf

Purpose: To exit from edm directly without being warned or queried.

r Request

Format: r newline

Purpose: Replace current line with newline.

Spacing: One space between r and newline is not significant. All other leading and embedded spaces become part of the text of the new line.

Pointer: Unchanged.

Default: If newline is null, a blank line replaces the current line.

s Request

Note: Used identically to the c request.

© Copyright, 1973, Massachusetts Institute of Technology
and Honeywell Information Systems Inc.

t Request

Format: t

Purpose: Moves pointer to the first line of the segment.

Pointer: At "no line" immediately above the first line of text.

Note: An i (insert) request immediately following a t request causes insertion of a text line at the beginning of the segment.

updelete Request

Format: updelete

Purpose: Delete all lines above (but not including) the current line.

Pointer: Unchanged.

upwrite Request

Format: upwrite path

Purpose: All the lines above the current line (but not including the current line) are saved in the hierarchy in the segment specified by path.

Spacing: A single space following upwrite is not significant.

Pointer: Unchanged.

Default: If path is not given, the name given in the invocation of edm is used.

Note: The lines written out are deleted from the edit buffers and thus are no longer available for editing.

© Copyright, 1973, Massachusetts Institute of Technology and Honeywell Information Systems Inc.

v Request

Format: v

Purpose: Causes edm to print responses following a c, f, l, n, or s request. This is the default mode.

Pointer: Unchanged.

Note: See k (kill) for inhibiting verbose mode.

w Request

Format: w path

Purpose: To write out (save) the edited copy. path can stipulate the directory and the entry name within the directory in which the segment is to be saved. If only the entry name for the saved copy is given, the working directory is assumed.

Spacing: A space between w and path is not significant.

Pointer: Set to "no line" at the end of the segment.

Default: If path is null, and if the original name of the segment is not null, the edited segment is saved under the original name; the original segment is deleted. If path is null and no previous segment exists, an error message is printed and edm looks for another request.

Note: To terminate editing without saving the edited copy, see the qf (quitforce) request.

© Copyright, 1973, Massachusetts Institute of Technology
and Honeywell Information Systems Inc. (END)

Name: exec_com, ec

The exec_com command is used to execute a series of command lines contained in a segment. It allows the user to construct command sequences that are invoked frequently without retyping the commands each time. In addition the segment can contain control statements that permit more flexibility than the simple execution of commands. Facilities exist for:

1. substitution of arguments to the command for special strings in the exec_com segment;

2. control of I/O streams;

3. generating command lines, control statements and input lines conditionally;

4. combining several exec_com sequences into one segment; and

5. altering the flow of control.

Usage

exec_com pathname -arg1- -arg2- ... -argn-

1) pathname — is the pathname of the segment containing the commands to be executed and control statements to be interpreted. The entry name of the segment must have the suffix ".ec", although the suffix can be omitted in the command invocation.

2) argi — is the string to be substituted for special strings in the exec_com segment.

The Input Segment

The exec_com segment should contain only command lines, input lines and control statements. It is normally created using a text editor, such as edm or qedx. The exec_com command can be used in conjunction with the abbrev command to form abbreviations for command sequences that are often used.

When the character "&" appears in the exec_com segment, it is interpreted as a special character. It is used to denote a string used for argument substution and to signify the start of a control statement.

© Copyright, 1973, Massachusetts Institute of Technology
and Honeywell Information Systems Inc.

Argument Substitution

Strings of the form "&i" in the exec_com segment are interpreted as dummy arguments and are replaced by the corresponding argument to the exec_com command. For instance, arg1 is substituted for the string "&1" and arg10 is substituted for "&10".

The character "&" should be followed by a number, i, or by the string "ec_name". If argi is not provided, "&i" is replaced by the null string. The string "&ec_name" is replaced by the entry name of the exec_com segment without the ".ec" suffix. the string "&0" is replaced by the pathname argument to exec_com, just as it was given to the command.

Argument substitution can take place in command lines, input lines or in control statements, since the replacement of arguments is done before the check for a control statement.

Control Statements

Control statements permit more variety and control in the execution of the command sequences. Currently there are twelve control statements: &label, &goto, &attach, &detach, &input_line, &command_line, &ready, &print, &if, &then, and &else.

Control statements generally must start at the beginning of a line with no leading blanks. Exceptions to this rule are the &then and &else statements, which can appear elsewhere. Also when a control statement is part of a THEN_CLAUSE or an ELSE_CLAUSE, it does not have to start at the beginning of a line.

1. &label and &goto

These statements permit the transfer of control within an exec_com segment.

&label location — identifies the place to which a goto control statement transfers control. location is any string of 32 or fewer characters identifying the label.

&goto location — causes control to be transferred to the place in the exec_com segment specified by the label location. Execution then continues at the line immediately following the label.

© Copyright, 1973, Massachusetts Institute of Technology and Honeywell Information Systems Inc.

2. &attach, &detach and &input_line

These statements allow the control of the I/O stream "user_input".

&attach	causes the I/O stream "user_input" to be attached to the exec_com segment. This means that if this control statement is executed, all input read by subsequent commands is taken from the segment rather from the stream "user_i/o".
&detach	causes the I/O stream "user_input" to be reverted to its original value. The default is detach rather than attach.
&input_line on	causes input lines returned when using the &attach feature to be written on the stream "user_output".
&input_line off	causes such input lines not to be written out. The default is on.

3. &command_line, &ready and &print

These statements allow the control of the I/O stream "user_output". They are useful as tools in observing the progress of the exec_com execution and in printing messages.

&command_line on	causes subsequent command lines to be written on the stream "user_output" before they are executed.
&command_line off	causes subsequent command lines not to be written out. The default is on.
&ready on	causes the invocation of the user's ready procedure after the execution of each command line.
&ready off	causes the user's ready procedure not to be invoked. The default is off.
&print char_string	causes the character string following &print to be written out on the I/O stream "user_output". The character "¬" is treated as a special character in an &print statement. The following is a list of strings that can appear and the characters

© Copyright, 1973, Massachusetts Institute of Technology and Honeywell Information Systems Inc.

that replace them:

string	replacement
¬/	new line character
¬|	form feed
¬-	horizontal tab
¬¬	¬

No other characters should appear following "¬" in the &print statement.

4. &quit

&quit causes exec_com to return to its caller and not to execute subsequent command lines.

5. &if, &then and &else

These statements provide the ability to generate command lines, input lines and control statements conditionally.

The form of these control statements is:

```
&if [ACTIVE_FUNCTION -arg1- ... -argn-]
&then THEN_CLAUSE
&else ELSE_CLAUSE
```

An active function in an &if control statement is evaluated. If the value of the active function is the string "true", THEN_CLAUSE is executed. If the value is "false", ELSE_CLAUSE is executed.

&if [ACTIVE_FUNCTION -arg1- ... -argn-], This statement must start at the beginning of a line. The active function is any active function (user-provided or system-supplied) that returns as its value a varying character string with the value "true" or "false". The arguments to the active function can themselves be active functions. (Nesting of active functions is permitted.) The active function and its optional arguments, enclosed in brackets, must be on the same line as the &if statement.

&then THEN_CLAUSE This statement must immediately follow the &if statement; it can appear on the same line or on the following line. THEN_CLAUSE is an exec_com statement, and can include a

© Copyright, 1973, Massachusetts Institute of Technology and Honeywell Information Systems Inc.

command line, an input line, the null statement and most control statements. The exceptions are &label, &if, &then and &else. (Nesting of &if statements is not permitted.) THEN_CLAUSE must be on the same line as the &then statement.

&else ELSE_CLAUSE — This statement is optional. When it appears it must immediately follow the &then statement; it can appear on the same line or on the following line. ELSE_CLAUSE is an exec_com statement, and can include a command line, an input line, the null statement and most control statements. The exceptions are &label, &if, &then and &else. ELSE_CLAUSE must be on the same line as the &else statement.

The active functions described in the MPM Reference Guide section, Logical Active Functions, are frequently used in the &if control statement.

Notes

If a line begins with the "&" character but is not one of the current control statements, the entire line is ignored. This is one way of including comments in the exec_com segment. The user is cautioned to leave a blank immediately following the "&" to insure compatibility with control requests to be added to exec_com in the future.

The segment executed by exec_com can contains calls to exec_com. The user is cautioned against frivolous use of this feature when using the &attach feature. When exec_com is called from an exec_com using this feature, the input read by commands in the second exec_com is read from the first exec_com segment. Generally if the &attach feature is used, all calls to exec_com should be preceded by &detach control statements.

Several exec_coms can be combined into one segment, by using the dummy argument "&ec_name" together with the &label and &goto statements. If exec_coms are grouped together, the exec_com segment should have all the names on its storage system entry that can replace "&ec_name" (concatenated with a ".ec" suffix).

© Copyright, 1973, Massachusetts Institute of Technology
and Honeywell Information Systems Inc.

Examples

1. Assume that the segment a.ec in the user's working directory contains:

```
pl1 &1 -table -list
dprint -dl &1.list
&quit
```

The command

```
exec_com a foo
```

would cause the following to be executed:

```
pl1 foo -table -list
dprint -dl foo.list
```

2. Assume that the segment b.ec in the user's working directory has an additional name a.ec and contains:

```
&goto &ec_name
&
&label b
print &1 1 99
&quit
&
&label a
pl1 &1 -table -list
dprint -dl &1.list
&quit
```

The command

```
exec_com b my_file
```

would cause the following to be executed:

```
print my_file 1 99
```

The command

```
exec_com a foo
```

would cause the following to be executed:

```
pl1 foo -table -list
dprint -dl foo.list
```

© Copyright, 1973, Massachusetts Institute of Technology
and Honeywell Information Systems Inc.

3. Assume that the segment d.ec in the user's working directory contains the following:

```
&if [exists segment &1.pl1] &then
&else &goto not_found
pl1 &1 -table -list
dprint -dl &1.list
&quit
&label not_found
&print &1.pl1 not found
&quit
```

If the segment foo.pl1 exists, the command

```
exec_com d foo
```

would cause the following to be executed:

```
pl1 foo -table -list
dprint -dl foo.list
```

If the segment foo.pl1 did not exist, the command

```
exec_com d foo
```

would output the following:

```
foo.pl1 not found
```

4. Assume that the segment test.ec in the user's working directory contains:

```
&print begin &ec_name exec_com
&command_line off
create &1.pl1
&command_line on
&attach
edm &1.pl1
i &1: proc;
&input_line off
i end &1;
w
q
&detach
&goto &2
&label compile
pl1 &1
&label nocompile
&print end &ec_name &1 &2 exec_com
```

© Copyright, 1973, Massachusetts Institute of Technology
and Honeywell Information Systems Inc.

The command

```
exec_com test x compile
```

produces the following output:

```
begin test exec_com
edm x.pl1
edit
i x: proc;

pl1 x
PL/1
end text x compile exec_com
```

© Copyright, 1973, Massachusetts Institute of Technology
and Honeywell Information Systems Inc. (END)*

Command
Standard Service System
12/9/71

Name: list, ls
listotals, lt
listnames, ln

The list command enables the user to determine the names (including multiple names), modes, times last used and modified, and lengths of either all the entries in a specified directory or selected entries. The related listotals and listnames commands give information only about counts and lengths, and names, respectively. All three commands present information about segments, directories, multi-segment files, and links (in that order, where appropriate), and respect various options as to searching and sorting, as explained below.

Usage: list, ls

list entry1 ... entryn -opt1- ... -optn-

1) entryi — is an entry name (see the -pn option for a directory). If no entries are specified, all the entries of the relevant directory are dealt with.

2) opti — is chosen from the following list of options (see also Notes below):

-pathname, -pn — uses the directory specified in the immediately following path name. The desired entries follow the path name. The default is the current working directory. -pn may occur more than once in a single invocation of the command. Entries preceding the first -pn refer to the working directory. (See Examples below.)

-segment, -sm — lists segments only. The default is on.

-directory, -dr — lists directories only. The default is off.

-multisegment_file
-msf — lists multi-segment files only. The default is off.

© Copyright, 1972, Massachusetts Institute of Technology
All rights reserved.

-link, -lk	lists links only. The default is off.
-branch, -br	lists branches, i.e., segments, directories, and multi-segment files only. The default is off.
-all, -a	lists all segments, directories, multi-segment files, and links. The default is off.
-date_time_used -dtu	sorts on and prints the date and time last used. The order is reverse chronological, i.e., the most recent first. The default is off.
-date_time_modified -dtm	sorts on and prints the date and time last modified. The order is reverse chronological, i.e., the most recent first. The default is off.
-reverse, -rv	reverses the order (to chronological or least recent first) for the -dtu or -dtm options. If -dtu is wanted, only -rv need be given. The default is off.

Notes

In the discussion of options, default means "what is assumed if this option is not given for a particular invocation of the command". on means the specified action is taken. off means the specified action is not taken.

The last three options (-dtu, -dtm, -rv) do not apply to links.

The star convention may be used in the entryi argument.

Conflicting options (e.g., -dtu and -dtm) should not be given in a single invocation of the command. Note that -sm, -dr, and -lk do not conflict.

© Copyright, 1972, Massachusetts Institute of Technology
All rights reserved.

Examples

```
list -a -rv
```

would list all the segments, directories, multi-segment files, and links in the current working directory, with the segments, directories, and multi-segment files sorted on the date and time last used, in chronological order.

```
list a b -pn >udd>Multics>Doe *.pl1 -pn >udd>MAC>Roe **
```

would list segments a and b (if present) in the working directory, all two component segments with a second component of pl1 in the directory >udd>Multics>Doe, and all of the segments in the directory >udd>MAC>Roe.

Usage: listotals, lt

```
listotals entry1 ... entryn -opt1- ... -optn-
```

1) entry*i* as above.

2) opt*i* as above except that the last three options are not valid (-dtu, -dtm, and -rv).

The response is a count of, and the total number of records occupied by, segments, directories, and multi-segment files, and a count of links, as appropriate to the particular options chosen.

Usage: listnames, ln

```
listnames entry1 ... entryn -opt1- ... -optn-
```

1) entry*i* as above.

2) opt*i* as above except that the last three options are not valid (-dtu, -dtm, and -rv).

The response is a list of the names of the segments, directories, multi-segment files, and links, as appropriate to the particular options chosen.

© Copyright, 1972, Massachusetts Institute of Technology
All rights reserved.

(END)

Name: mail, ml

The mail command allows the user to send a segment to another user or to print messages sent to him. Mail sent to a user is placed in the segment "mailbox" in his home directory. The mailbox is provided with a lock.

Usage

mail -path- -person1- -project1- ... -personn- -projectn-

1) path — is the pathname of a mailbox segment to be printed (if no personi projecti pairs are supplied) or of a segment to be sent to other user or users (when one or more personi projecti pairs are specified).

2) personi — is the name of a person to whom mail is to be sent.

3) projecti — is the name of a project on which personi works.

Printing Mail

The mailbox segment named by path will be locked and its contents printed out preceded by a line of the form

x messages, y lines

If path is omitted, the segment "mailbox" in the user's home directory is printed. After the mail is printed, the mail command will ask whether to delete the mail. If the answer is no, the command mail unlocks the mailbox and terminates it; if the answer is yes, mail truncates the mailbox to zero length.

Sending Mail

The contents of the segment path will be copied into the segment

>user_dir_dir>projecti>personi>mailbox

for each person-project pair specified. When mail is sent, the command first locks the target mailbox and then copies the contents of the mail into the target mailbox, preceded by a header identifying the sender:

From Person.Project date time

© Copyright, 1972, Massachusetts Institute of Technology
All rights reserved.

The segment to be mailed must be less than one record long. Illegal characters will be removed from the mail as it is sent.

If path is *, mail will respond "Input" and accept lines from the keyboard until a line with only a period (.) is typed; the typed lines then will be sent to the specified addresses.

Note

The user of the mail command must have rwa access to any mailbox he accesses, and e access to the directory superior to the mailbox in order to read or send mail.

Entry: mail$unlock

The user will receive a message of the form

```
mail:  The segment is already locked.
       mailbox busy, try again later.
```

if the mailbox he attempts to access is locked. Because quitting out of mail may leave a mailbox locked, this entry is provided to force the lock to zero. The mailbox specified by path will have its lock cleared. If path is not specified, the mailbox in the user's home directory will be unlocked.

Usage

```
mail$unlock -path-
```

1) path as above.

Creating a Mailbox Segment:

In order to receive mail, there must be a segment in the user's home directory called mailbox whose ACL is set to rwa for all users. To create such a segment perform the following sequence of commands in your home directory.

```
create mailbox
setacl mailbox rwa *.*.*
```

© Copyright, 1972, Massachusetts Institute of Technology
All rights reserved.

(END)

Name: qedx, qx

The qedx context editor can be used to create and edit ASCII segments in Multics. The editor is based on the QED editor as implemented by K. L. Thompson of Bell Telephone Laboratories. qedx is basically a subset of QED and is designed to provide the user with much of the power of QED and performance similar to the simpler edm editor also described in this manual.

Usage

qedx -inst_path- -arg1- ... -argn-

1. inst_path — is an optional argument and, if present, specifies the pathname of an ASCII segment from which the editor is to take its initial instructions. Such a set of instructions is commonly referred to as a macro. The editor automatically concatenates the suffix ".qedx" to inst_path to obtain the complete pathname of the segment containing the qedx instructions.

2. argi — is an optional argument that is appended as a separate line to the buffer "args"; arg1 becomes the first line in the buffer, and argn becomes the last. Arguments are used in conjunction with a macro specified by instr_path. See Initalization of Macros below.

If inst_path is provided, the editor executes the qedx requests contained in the specified segment and then interrogates the user's terminal for further requests. If inst_path is omitted, the editor immediatesly interrogates the user's terminal for the first qedx request. The use of the inst_path argument requires a fairly detailed understanding of the editor and further discussion of this feature is delayed until later in this write-up. For the moment, we will restrict our discussion to the most straightforward use of the editor.

Once the qedx command is invoked, the user can immediately begin to issue qedx requests from his terminal. Requests fall into one of two general categories, input requests and edit requests. Input requests place the editor into input mode, which allows the user to enter new ASCII text from his terminal until an appropriate escape sequence is typed to switch the editor back

© Copyright, 1973, Massachusetts Institute of Technology
and Honeywell Information Systems Inc.

CB1 61 Sor 2

to edit mode. Edit requests allow the user to read and write ASCII segments and perform various editing functions on ASCII data. Input and editing operations are not performed directly on the target segments but in a temporary workspace known as a buffer.

To create a new ASCII segment, a user might perform the following steps.

1. Invoke qedx and enter input mode by typing one of the input requests (e.g., append) as the first qedx request.

   1a) Enter ASCII text lines into the buffer from the terminal.

   1b) Leave input mode by typing the appropriate escape sequence as the first characters of a new line.

2. Inspect the contents of the buffer and make any necessary corrections using edit or input requests.

3. Write the contents of the buffer into a new segment using the write request and exit from the editor using the quit request.

To edit an existing ASCII segment, a user might perform the following steps.

1. Invoke qedx and read the segment into the buffer by giving a read request as the first qedx request.

2. Edit the contents of the buffer using edit and input requests as necessary.

3. Using the write request, write the contents of the modified buffer either back into the original segment or, perhaps, into a segment of a different name and exit from the editor.

The user can create and edit any number of segments with a single invocation of the editor as long as the contents of the buffer are deleted before work is started on each new segment.

Editor Requests

In the list given below, editor requests are divided into three categories: input requests, basic edit requests, and extended edit requests. The basic edit requests are sufficient to allow a user to create and edit ASCII segments and provide a

© Copyright, 1973, Massachusetts Institute of Technology
and Honeywell Information Systems Inc.

functional capability quite similar to edm. The extended requests are, in general, a little more difficult to learn to use and the discussion of these requests is postponed until later in this write-up. Note that the letter given in parentheses is the actual character used to invoke the request in qedx and does not always bear a relation to the name of the request.

1. Input Requests

append (a)	Enter input mode, append lines typed from the terminal after a specified line.
change (c)	Enter input mode, replace the specified line or lines with lines typed from the terminal.
insert (i)	Enter input mode, insert lines typed from the terminal before a specified line.

2. Basic Edit Requests

delete (d)	Delete specified line or lines from the buffer.
print (p)	Print specified line or lines on the terminal.
quit (q)	Exit from the editor.
read (r)	Read specified segment into the buffer.
substitute (s)	Replace specific character strings in specified line or lines.
write (w)	Write current buffer into specified segment.

3. Extended Edit Requests

execute (e)	Pass remainder of request line to the Multics command processor (i.e., escape to other commands).
line number (=)	Print line number of specified line.
global (g)	Print, delete or print line number of all addressed lines that contain a specified character string.

© Copyright, 1973, Massachusetts Institute of Technology and Honeywell Information Systems Inc.

exclude (v)	Print, delete or print line number of all addressed lines that do not contain a specified character string.
buffer (b)	Switch to specified buffer (i.e., change base of all subsequent editor operations to specified buffer).
move (m)	Move specified line or lines into a specified auxiliary buffer.
status (x)	Print on the terminal a summary of the status of all buffers currently in use.

Addressing

The qedx editor is basically a line-oriented editor in that editing requests usually operate on an integral number of ASCII lines. As a result, most editing requests are preceded with an address specifying the line or lines in the buffer on which the request is to operate. There are three basic means by which lines in the buffer can be addressed:

1. Addressing by line number;

2. Addressing relative to the "current" line;

3. Addressing by context.

In addition, a line address can be formed using a combination of the above techniques.

1. Addressing by Line Number

Each line in the buffer can be addressed by a decimal number indicating the current position of the line within the buffer. The first line in the buffer is line 1, the second line 2, etc. The last line in the buffer can be addressed either by line number or by using the $ character, which is interpreted to mean "the last line currently in the buffer". In certain cases it is possible to address the (fictitious) line preceding line 1 in the buffer by addressing line 0.

As lines are added to or deleted from the buffer, the line numbers of all lines that follow the added or deleted lines are relocated accordingly. For example, if line 15 is deleted from the buffer, line 16 becomes line 15, 17 becomes 16, and so on.

© Copyright, 1973, Massachusetts Institute of Technology
and Honeywell Information Systems Inc.

If an attempt is made to address a line not contained in the buffer, an error message is printed by the editor. If the buffer is currently empty, as it is when the editor is first entered, only the line numbers 0 and $ are considered valid.

2. Addressing Relative to the Current Line

The qedx editor maintains the notion of a "current" line that is addressable by using the character "." (period) to represent the address of the current line. Normally, the current line is the last line addressed by an edit request or the last line entered from the terminal by an input request. The value of "." after each editor request is documented in the description of the request.

Lines can be addressed relative to the current line number by using an address consisting of "." followed by a signed decimal number specifying the position of the desired line relative to the current line. For example, the address .+1 specifies the line immediately following the current line and the address .-1 specifies the line immediately preceding the current line.

When specifying an increment to the current line number, the + sign can be omitted (e.g., .5 is interpreted as .+5.. In addition, when specifying a decrement to the current line number, the "." itself can be omitted (e.g., -3 is interpreted as .-3.. It is also possible to follow "." with a series of signed decimal numbers (e.g., .5+5-3 is interpreted as .+7).

3. Addressing by Context

Lines can be addressed by context by using a regular expression to match a string of characters on a line. When used as an address, a regular expression specifies the first line encountered that contains a string of characters that matches the regular expression. In its simplest form, a regular expression used as an address performs in a manner similar to the edm locate (l) request. For example, in the following text, the regular expression /abc/ matches line 2.

```
a:  procedure;
    abc=def;
    x=y;
    end a;
```

© Copyright, 1973, Massachusetts Institute of Technology
and Honeywell Information Systems Inc.

To use a regular expression as an address, the user types /regexp/, where regexp is any valid regular expression as described below. The search for a regular expression begins on the line following the current line (i.e., .+1. and continues through the entire buffer, if necessary, until it again reaches the current line. In other words, the search proceeds from .+1 to $ and then from line 1 to the current line. If the search is successful, /regexp/ specifies the first line encountered during the search in which a match was found.

A regular expression can consist of any character in the ASCII set except the new line character. However, the following characters have specialized meanings in regular expressions.

/ Delimits a regular expression used as an address.

* Signifies "any number (or none) of the preceding character".

¬ When used as the first character of a regular expression, the ¬ character signifies the character preceding the first character on a line.

$ When used as the last character of a regular expression, the $ character signifies the character following the last character on a line.

. Matches any character on a line.

Some examples follow:

/a/	Matches the letter "a" anywhere on a line.
/abc/	Matches the string "abc" anywhere on a line.
/ab*c/	Matches "ac", "abc", "abbc", "abbbc", etc. anywhere on a line.
/in..to/	Matches a line containing "in" followed by any two characters followed by "to".
/in.*to/	Matches a line containing "in" and "to" in that order.
/¬abc/	Matches a line beginning with "abc".

© Copyright, 1973, Massachusetts Institute of Technology
and Honeywell Information Systems Inc.

/abc$/	Matches a line ending with "abc".
/¬abc.*def$/	Matches a line beginning with "abc" and ending with "def".
/.*/	Matches any line.
/¬$/	Matches an empty line (new line character only).

The special meanings of "/", "*", "$", "¬", and "." within a regular expression can be removed by preceding the special character with ¢c.

/¢c/¢c*/	Matches the string "/*" anywhere on a line.

The editor remembers the last regular expression used in any context. The user can reinvoke the last used regular expression by using a null regular expression (e.g., //). In addition, a regular expression can be followed by a signed decimal integer in the same manner as when addressing relative to the current line number. For example, the addresses /abc/+5-3, /abc/+2 or /abc/2 all address the second line following a line containing "abc".

Note that the two uses of "." and "$" (as line numbers and as special characters in regular expressions) are distinguished by context.

4. Compound Addresses

An address can be formed using a combination of the techniques described above. The following rules are intended as a general guide in the formation of these compound addresses.

1. If a line number is to appear in an address, it must be the first component of the address.

2. A line number can be followed by a regular expression. This construct is used to begin the regular expression search after a specific line number. For example, the address 10/abc/ starts the search for /abc/ immediately after line 10.

© Copyright, 1973, Massachusetts Institute of Technology and Honeywell Information Systems Inc.

3. A regular expression can follow an address specified relative to the current line number. For example, the address .-8/abc/ starts the search from 8 lines preceding the current line.

4. A regular expression can be followed by another regular expression. For example, the address /abc//def/ matches the first line containing "def" appearing after the first line containing "abc". As mentioned earlier, a regular expression can be followed by a decimal integer. For example, the address /abc/-10/def/.5 starts the search for /def/ from 10 lines preceding the first line to match /abc/ and if /def/ is matched, the value of the compound address is the fifth line following the line containing the match for /def/.

5. Addressing a Series of Lines

Several of the editor requests can be used to operate on a series of lines in the buffer. To specify a series of lines, two addresses must be given in the following general form:

A1,A2

The pair of addresses specifies the series of lines starting with the line addressed by the address A1 through the line addressed by A2 inclusive.

Examples

1,5	specifies line 1 through line 5.
1,$	specifies the entire contents of the buffer.
.1,/abc/	specifies the line following the current line through the first line after the current line containing "abc".

When a comma is used to separate addresses, the address computation of the second address is unaffected by the computation of the first address (i.e., the value of "." is not changed by the evaluation of the first address). For example, the address pair

.1,.2

specifies a series of two lines, the line immediately after the current line through the second line after the current line.

© Copyright, 1973, Massachusetts Institute of Technology
and Honeywell Information Systems Inc.

However, if a semicolon is used to separate addresses instead of a comma, the value of "." is set to point to the line addressed by A1 before the evaluation of A2 begins. In contrast to the example given immediately above, the address pair

```
.1;.2
```

specifies a series of three lines, the line immediately following the original current line through the second line following the line specified by A1. As a further example, the address pair

```
/abc/;.+10
```

is equivalent to the address pair

```
/abc/,/abc/+10
```

6. Addressing Errors

The following list describes the various errors that can occur when the editor is attempting to evaluate an address.

1. "Buffer empty" - An attempt has been made to reference a specific line when the buffer is empty. (Only "$", "." and "0" are legal addresses within an empty buffer and only if used with a read, append, or insert request.)

2. "Address out of buffer" - An attempt has been made to refer to a nonexistent line (e.g., an address of 20 when there are fewer than 20 lines in the buffer or an address of .+5 when the current line is fewer than 5 lines from the last line in the buffer).

3. "Address wrap around" - An attempt has been made to address a series of lines in which the line number of the second line addressed is less than the line number of the first (e.g., $,1).

4. "Search failed" - A regular expression search initiated from the user's terminal has failed to find a match.

5. "Syntax error in regular expression" - A regular expression used as an address has not been properly delimited.

© Copyright, 1973, Massachusetts Institute of Technology
and Honeywell Information Systems Inc.

6. "// undefined" - A null regular expression has been used and no previously typed regular expression is available.

Use of the Editor

1. Editor Request Format

A request to the editor can take any one of the following three forms depending on the number of addresses to be specified with the request.

1. <request>

2. ADR<request>

3. ADR1,ADR2<request> or ADR1;ADR2<request>

ADR, ADR1, and ADR2 are any legal addresses as specified above, and <request> is any valid editor request.

Some editor requests require no address, some require a single address and others require a pair of addresses. In all cases, however, the user can use a request omitting one or both of the required addresses and let the editor provide the missing address information by default. The following general rules apply to the use of addresses specified by default.

1. If a request requiring an address pair is issued with the second address missing, the (missing) second address is assumed to be the same as the first. For example,

   ADR<request>

   is interpreted as

   ADR,ADR<request>

   and addresses a single line in the buffer (i.e., the line addressed by ADR).

2. If a request requiring an address pair is issued with both addresses missing, one of the following address pairs is assumed depending on the request issued.

   .,.<request> for most editor requests

   1,$<request> for write, global and exclude

© Copyright, 1973, Massachusetts Institute of Technology and Honeywell Information Systems Inc.

3. If a request requiring a single address is issued with no address specified, one of the following addresses is assumed depending on the request issued.

   .<request> for most editor requests

   $<request> for read requests

2. The Value of "."

All editor requests that alter the contents of the buffer or cause information to be output on the user's terminal change the value of "." (i.e., the current line). Usually, the value of "." is set to the last address specified (either explicitly or by default) in the editor request. The one major exception to this rule is the delete request, which sets "." to the line after the last line deleted.

3. Multiple Requests on a Line

In general, any number of editor requests can be issued in a single input line. However, each of the requests listed below must be terminated with a new line character, and, thus, each must appear on a line by itself or at the end of a line containing multiple editor requests.

read	(r)
write	(w)
quit	(q)
execute	(e)

4. Spacing

The following rules govern the use of spaces in editor requests.

1. Spaces are taken as literal text when appearing inside of regular expressions. Thus, /the n/ is not the same as /then/.

2. Spaces cannot appear in numbers, i.e., 13 cannot be written as 1 3 (which is interpreted as 1+3 or 4..

3. Spaces within addresses except as indicated above are ignored.

© Copyright, 1973, Massachusetts Institute of Technology
and Honeywell Information Systems Inc.

4. The treatment of spaces in the body of an editor request depends on the nature of the request.

5. Comments

The quotation mark character is reserved as the comment delimiter and is actually implemented as an editor request, the effect of which is to ignore the remainder of the request line. If the quotation mark is preceded by an address, the value of "." is set to that address.

6. The Locate Request

If an address is followed by a new line character, the value of "." is set to the addressed line and the line is printed on the user's terminal. For example, the request line

/¬start/

locates a line beginning with "start", sets the value of "." and prints the line.

7. Responses from the Editor

In general, the editor does not respond with output on the terminal unless explicitly requested to do so (e.g., with a print or print line number request). The editor does not comment when the user enters or exits from the editor or changes to and from input and edit modes. The use of frequent print requests is recommended for users using the qedx editor for the first time.

If the user inadvertently requests a large amount of terminal output from the editor and wishes to abort the output without abandoning the edit, he can hit the quit button on his terminal, and, after the quit response, he can re-enter the editor by invoking the program_interrupt (pi) command. This action causes the editor to abandon its printout, but leaves the value of "." as if the printout had gone to completion.

If an error is encountered by the editor, an error message is printed on the user's terminal and any editor requests already input (i.e., read ahead from the terminal) are discarded.

If a user exits from qedx by hitting the quit button, and subsequently invokes qedx in the same process, the message "qedx: Pending work in previous invocation will be lost if you proceed; do you wish to proceed?" is printed on the terminal. The user must type a "yes" or "no" answer.

© Copyright, 1973, Massachusetts Institute of Technology
and Honeywell Information Systems Inc.

8. Input Mode

The editor can be placed in input mode with the use of one of the three input requests (append, change and insert). The input request must immediately be followed by a blank or a new line character, which in turn is followed by the literal text to be input to the buffer. The literal text can contain any number of ASCII lines. To exit from input mode and terminate the input request, the escape sequence ¢f is typed, usually as the first character of a new line. The ¢f escape sequence can be followed immediately with more editor requests on the same line. The usual form of an input request is as follows:

```
ADR1(,ADR2.<input request>
TEXT
...
¢f
```

It is important to remember to terminate the input request with the ¢f before typing another request. Otherwise, the (would be) editor request is regarded as input and included in the text rather than executed as a request.

Upon leaving input mode, the value of "." is set to point at the last line input from the terminal.

The special meaning of any of the escape sequences used by qedx (e.g., ¢f, ¢c, ¢b, ¢r) can be suppressed by preceding the escape sequence with ¢c thus allowing these escape sequences to be input as literal text.

Input Requests

1. Append (a)

The append request is used to enter input lines from the terminal, appending these lines after the line addressed by the append request. The append request is one of the few requests that can operate correctly when the buffer is empty.

Format:
```
ADRa
TEXT
¢f
```

Default: a is taken to mean .a

© Copyright, 1973, Massachusetts Institute of Technology
and Honeywell Information Systems Inc.

Value of ".": Set to last line appended.

Example: Before

```
     a:  procedure;
         x=y;
         end a;
```

Request Sequence

```
     2a                    /x=/a
     q=r;       or         q=r;
     ¢f                    ¢f
```

After

```
     a:  procedure;
         x=y;
"."->    q=r;
         end a;
```

Note: The request 0a can be used to insert text before line 1 of the buffer.

2. Change (c)

The change request is used to delete an addressed line or set of lines and replace the deleted line(s) with new text entered from the terminal.

Format:
```
ADR1,ADR2c
TEXT
¢f
```

Default: c is taken to mean .,.c

Value of ".": Set to last line entered from terminal.

Example Before

```
     a:  procedure;
         x=y;
         q=r;
         end a;
```

© Copyright, 1973, Massachusetts Institute of Technology and Honeywell Information Systems Inc.

Request Sequence

```
2,3c                    /x=/,/q=/c
s=t;          or        s=t;
u=v;                    u=v;
w=z;                    w=z;
¢f                      ¢f
```

After

```
    a:  procedure;
        s=t;
        u=v;
"."->   w=z;
        end a;
```

3. Insert (i)

The insert request is used to enter input lines from the terminal and insert the new text immediately before the addressed line. The insert request is one of the few requests that can operate on an empty buffer.

Format:

```
ADRi
TEXT
¢f
```

Default: i is taken to mean .i

Value of ".": Set to last line inserted.

Example Before

```
    a:  procedure;
        x=y;
        end a;
```

Request Sequence

```
2i                      /x=/i
q=r;          or        q=r;
¢f                      ¢f
```

© Copyright, 1973, Massachusetts Institute of Technology
and Honeywell Information Systems Inc.

After

```
      a:  procedure;
"."-> q=r;
      x=y;
      end a;
```

Note: The request ADRi has the same effect as the request ADR-1a.

Basic Edit Requests

The edit requests described below represent a subset of qedx suitable for most editing situations. Additional requests are described later in this write-up under extended editor functions.

1. Delete (d)

The delete request is used to delete the addressed line or set of lines from the buffer.

Format: ADR1,ADR2d

Default: d is taken to mean .,.d

Value of ".": Set to line immediately following the last line deleted.

Example: Before

```
     a:  procedure;
         x=y;
         q=r;
         s=t;
         end a;
```

Request Sequence

```
     3,4d    or     /q=/,/s=/d
```

After

```
     a:  procedure;
         x=y;
"."-> end a;
```

© Copyright, 1973, Massachusetts Institute of Technology
and Honeywell Information Systems Inc.

2. Print (p)

The print request is used to print the addressed line or set of lines on the user's terminal.

Format: ADR1,ADR2p

Default: p is taken to mean .,.p

Value of ".": Set to last line addressed by the print request (i.e., the last line to be printed).

Example: Contents of Buffer

```
     a:  procedure;
         x=y;
         q=r;
         s=t;
         end a;
```

Request

```
     2,4p     or      /x=/,/s=/p
```

Console Output

```
          x=y;
          q=r;
"."->     s=t;
```

Note: The output from the print request can be aborted with the use of the quit button and program_interrupt command.

3. Quit (q)

The quit request is used to exit from the editor and does not itself save the results of any editing that might have been done. If the user wishes to save the modified contents of the buffer, he must explicitly issue a write request (see below).

Format: q

Default: The quit request cannot have an address.

Special Note: The quit request must be followed immediately by a new line character.

© Copyright, 1973, Massachusetts Institute of Technology
and Honeywell Information Systems Inc.

4. Read (r)

The read request is used to append the contents of a specified ASCII segment after the addressed line. The read request is one of the few requests that operate correctly when the buffer is empty.

Format: ADRr PATH

PATH is the pathname of the ASCII segment to be read into the buffer. The pathname can be preceded with any number of spaces and must be followed immediately by a new line character.

Default: r PATH is taken to mean $r PATH

Value of ".": Set to the last line read from the file.

Example: Before

```
     a:  procedure;
         x=y;
         end a;
```

Request

```
     2r b.pl1   or   /x=/r b.pl1
```

where b.pl1 is the following text

```
     b:  procedure;
         c=d;
         end b;
```

After

```
     a:  procedure;
         x=y;
     b:  procedure;
         c=d;
"."->    end b;
         end a;
```

Note: The request 0r PATH is used to insert the contents of a segment before line 1 of the buffer.

© Copyright, 1973, Massachusetts Institute of Technology
and Honeywell Information Systems Inc.

5. Substitute (s)

The substitute request is used to modify the contents of the addressed line or set of lines by replacing all strings that match a given regular expression with a specified character string.

Format: ADR1,ADR2s/REGEXP/STRING/

(The first character after the "s" is taken to be the request delimiter and can be any character not appearing either in REGEXP or in STRING.)

Default: s/REGEXP/STRING/ is taken to mean .,.s/REGEXP/STRING/

Value of ".": Set to last line addressed by request.

Operation: Each character string in the addressed line or lines that matches REGEXP is replaced with the character string STRING. If STRING contains the special character &, each & is replaced by the string matching REGEXP. The special meaning of & can be suppressed by preceding the & with the escape sequence ¢c.

Examples:

```
Before:     The quick brown sox
Request:    s/sox/fox/
After:      The quick brown fox

Before:     xyzindex=q;
Request:    s/index/(&)/
After:      xyz(index)=q;

Before:     x=y
Request:    s/$/;/
After:      x=y;
```

6. Write (w)

The write request is used to write the addressed line or set of lines into a specified segment. If the segment does not already exist, a new segment is created with the specified name.

© Copyright, 1973, Massachusetts Institute of Technology
and Honeywell Information Systems Inc.

Format: ADR1,ADR2w -PATH-

PATH is the pathname of the segment the contents of which are to be replaced by the addressed lines in the buffer. The pathname can be preceded by any number of spaces and must be followed immediately by a new line character. If PATH is omitted, If PATH is omitted and the default pathname is null, then the message "No pathname given" is printed and qedx awaits another request. The default pathname is the first pathname used with either a read or write request in this invocation of qedx. The default pathname is set to null if no pathname has been given or if a pathname (either the same one or a different one) is used with a read or write request a second time.

Default: w PATH is taken to mean 1,$w PATH

Value of ".": Unchanged

Example: Request

```
2,4w sam.pl1
```

Result

The second through fourth lines of the buffer replace the contents of the segment sam.pl1 in the user's working directory.

Extended Use of the Editor

The editor requests discussed up to this point comprise a basic subset sufficient for most applications. A user learning to use qedx for the first time might be well advised to stop at this point.

The editor requests remaining to be described are divided into two groups: those that require a knowledge of auxiliary buffers and those that do not. Discussion of the former group of requests is delayed until the section on auxiliary buffers. Discussion of the latter group of requests proceeds below.

© Copyright, 1973, Massachusetts Institute of Technology
and Honeywell Information Systems Inc.

1. Execute (e)

The execute request is used to invoke the Multics command system without exiting from the editor. Whenever an execute request is recognized, the remaining characters following the execute request in the request line are passed to the Multics command processor. The execute request can be followed by any legal Multics command line. However, due to a temporary restriction, the user should not invoke edm, bsys, or qedx while in qedx.

Format: e <command line>

Value of ".": Unchanged

Example: The request line

```
e print alpha.pl1
```

can be used to print a segment in the user's working directory.

The request line

```
e list; mail
```

lists the user's working directory and prints his mailbox (if any).

Note: If the user wishes to abort a command line issued with the execute request, he can hit quit and invoke the program_interrupt command to abort the command line and restore control to qedx.

2. Print Line Number (=)

There request is used to print the line number of the addressed line.

Format: ADR=

Default: = is taken to mean .=

Value of ".": Set to line addressed by request

© Copyright, 1973, Massachusetts Institute of Technology and Honeywell Information Systems Inc.

Example: Contents of Buffer

```
a:  procedure;
    x=y;
    p=q;
    end a;
```

Request

```
/q;/=
```

Response

```
3
```

3. Global (g)

The global request is used in conjunction with some other request (e.g., print, delete, print line number). That request is to operate only on those lines addressed by the global request that contain a match for a specified regular expression.

Format: ADR1,ADR2gX/REGEXP/

where X must be one of the following requests

d	delete lines containing REGEXP
p	print lines containing REGEXP
=	print line numbers of lines containing REGEXP

Default: gX/REGEXP/ is taken to mean 1,$gX/REGEXP/

Value of ".": Set to ADR2 of request

Note: The character immediately following the request X is taken to be the regular expression delimiter and can be any character not appearing in REGEXP.

Example: Before

```
a:  procedure;
    q=r;
    x=y;
    y=q;
    end a;
```

© Copyright, 1973, Massachusetts Institute of Technology
and Honeywell Information Systems Inc.

Request

```
1,$gd/q/
```

After

```
     a:  procedure;
         x=y;
"."->    end a;
```

4. Exclude (v)

The exclude request is also used in conjunction with another request (e.g., print, delete, print line number). That request is to operate only on those lines addressed by the exclude request that do not contain a match for a specified regular expression.

Format: ADR1,ADR2vX/REGEXP/

where X must be one of the following requests

d	delete lines not containing REGEXP
p	print lines not containing REGEXP
=	print line numbers of lines not containing REGEXP

Default: vX/REGEXP/ is taken to mean 1,$vX/REGEXP/

Value of ".": Set to ADR2 of request

Note: The character immediately following the request X is taken to be the regular expression delimiter and can be any character not appearing in REGEXP.

Example: Before

```
     a:  procedure;
         q=r;
         x=y;
         y=q;
         end a;
```

Request

```
1,$v=/q/
```

© Copyright, 1973, Massachusetts Institute of Technology
and Honeywell Information Systems Inc.

Response

1
3
5

Auxiliary Buffers

The discussion up to this point has assumed the existence of only a single buffer. Actually, qedx supports a virtually unlimited number of buffers. One buffer at a time can be designated as the "current buffer"; any other buffers at this time are referred to as auxiliary buffers. All of the editor requests described so far operate within the current buffer.

Each buffer is given a symbolic name of 1 to 16 ASCII characters. When the editor is invoked, a single buffer (buffer 0) is created by the editor and designated as the current buffer. Additional buffers can be created merely by referencing a previously undefined buffer name. Each buffer is implemented as a separate segment in the user's process directory and, thus, is capable of holding any ASCII segment.

Buffer names are usually enclosed in parentheses; for example, the buffer name Fred is typed as (Fred). However, for historical reasons, a buffer name consisting of a single character can be typed with or without the enclosing parentheses (e.g., "x" is taken to be "(x)").

1. The Change Buffer Request (b)

The change buffer request is used to designate an auxiliary buffer as the current buffer. The previously designated current buffer becomes an auxiliary buffer.

Format: b(X)

where X is the name of the buffer that is to become the current buffer.

Value of ".": Restored to the value of "." when buffer X was last used as the current buffer (i.e., the value of "." is maintained separately for each buffer and saved as part of the buffer status).

© Copyright, 1973, Massachusetts Institute of Technology
and Honeywell Information Systems Inc.

2. The Move Request (m)

The move request is used to move one or more lines from the current buffer to a specified auxiliary buffer. The addressed lines replace the previous contents (if any) of the auxiliary buffer.

Format: ADR1,ADR2m(X)

where X is the name of the auxiliary buffer to which the lines are to be moved.

Default: m(X) is taken to mean .,.m(X)

Value of ".": Set to first line after last line moved in current buffer. Set to line 0 in the specified auxiliary buffer.

Example:

Before:

	Current Buffer	Buffer B
a:	procedure;	abc=def;
	x=y;	end bin;
	y=k;	
	k=r;	
	end a;	

Request: 3,4m(B) or /k;/,/r;/m(B)

After:

	Current Buffer	Buffer B
a:	procedure;	y=k;
	x=y;	k=r;
	end a;	

3. The Buffer Status Request (x)

The buffer status request is used to print on the user's terminal a summary of the status of all buffers currently in use. The name and length (in lines) of each buffer is listed; the current buffer is specially marked with a right arrow "->" immediately to the left of the buffer name.

Format: x

Value of ".": Unchanged

© Copyright, 1973, Massachusetts Institute of Technology
and Honeywell Information Systems Inc.

Example: If the user has created the additional buffers alpha and beta and has designated alpha as his current buffer, the output from the buffer status request might be as follows.

```
157     (0)
 32   ->(alpha)
 53     (beta)
```

This output indicates 157 lines in buffer 0 (the initial buffer), 32 lines in alpha (the current buffer) and 53 lines in beta.

4. Special Escape Sequences

The input to qedx can be viewed as a stream of ASCII characters. Depending on the context, some of these characters are interpreted as editor requests and others are interpreted as literal text. The following escape sequences are recognized by the editor, in either context, as directives to alter the input character stream in some fashion.

¢b(X) This sequence is used to redirect the editor input stream to read subsequent input from buffer X. When the editor encounters this sequence, the entire escape sequence is removed from the input stream and replaced with the literal contents of the specified buffer. If another ¢b escape sequence is encountered while accepting input from buffer X, the newly encountered escape sequence is also replaced by the contents of the named buffer. The editor allows the recursive replacement of ¢b escape sequences by the contents of named buffers to a recursion depth of 500 nested ¢b escape sequences.

The buffer to which the input stream is redirected can contain editor requests, literal text or both. If the editor is executing a request obtained from a buffer (rather than from the terminal) and the request specifies a regular expression search for which no match is found, the usual error comment is suppressed and the remaining contents of the buffer are skipped. If one thinks of the escape sequence ¢b(X) as a subroutine call statement, the failure to match a regular expression specified by some request in buffer X can be thought of as a return statement.

© Copyright, 1973, Massachusetts Institute of Technology
and Honeywell Information Systems Inc.

¢r This escape sequence is used to temporarily redirect the input stream to read a single line from the user's terminal and is normally used when executing editor requests contained in a buffer. The ¢r is removed from the input stream and replaced with the next complete line entered from the user's terminal. In the line that replaces the ¢r sequence, additional ¢r or ¢b escape sequences have no effect.

Note: The special meanings of ¢b and ¢r can be suppressed by preceding the escape sequence with a ¢c escape sequence.

5. Use of Buffers for Moving Text

Perhaps the most common use of buffers in qedx is for moving text from one part of a segment to another. A typical pattern is to move the text to be moved into an auxiliary buffer with a move request. For example, the request

```
1,5m(temp)
```

moves line 1 through 5 of the current buffer into the auxiliary buffer temp. Once the lines have been moved to an auxiliary buffer, they can be used as literal text in conjunction with an input request. For example, to insert the lines in buffer temp immediately before the last line in the current buffer, the following sequence might be used.

```
$i
¢b(temp)¢f
```

In this case, the literal text in buffer temp replaces the ¢b escape sequence and thus is treated as input to the editor already placed in input mode by the insert (i) request. Notice that the ¢f immediately following the ¢b escape sequence is correct since it can be expected that the last line in buffer temp is terminated by a new line character that precedes the ¢f after the ¢b(temp) is expanded.

6. Defining Editor Cliches

Another common use for buffers is for the definition of frequently used editing sequences or "cliches". For example, if a programmer were faced with the task of adding the same source code sequence in several places in a program, he might elect to type the editing sequence into a buffer only once and then invoke

© Copyright, 1973, Massachusetts Institute of Technology
and Honeywell Information Systems Inc.

the contents of the buffer as many times as necessary. In the example given below, the contents of buffer new contains the necessary editor requests and literal text to append four lines of text at any point in the current buffer.

Example: Contents of buffer new

```
a
if code ¬=0 then do;
   call error (code);
   return;
   end;
¢f
.-4,.1p
```

Usage

```
ADR¢b(new)
```

ADR becomes the address of the append request in buffer new and specifies the point at which the literal text is to be appended. The four lines of text in buffer new (lines 2-5. are appended to the current buffer, the ¢f terminates the append request and the print request prints the line preceding the appended lines, the four appended lines and the line following the appended lines.

7. Use of Editor Macros

The use of buffers in qedx allows a user to place more elaborate editor request sequences (commonly called macros) into auxiliary buffers and use the editor as a pseudo-programming language. In this context, it is useful to regard a buffer containing executable editor requests as a subroutine and to view the ¢b escape sequence as a call statement.

In the example discussed below, a macro is implemented to read ASCII text from the terminal until an input terminating sequence, a line consisting only of ".", is typed. When the terminating sequence is typed, the macro asks the user for a name under which the input is filed and exits from the editor. The macro is implemented with two executable buffers (subroutines) named read and test and is invoked by diverting the input stream to the read buffer (i.e., by calling the read subroutine).

© Copyright, 1973, Massachusetts Institute of Technology
and Honeywell Information Systems Inc.

Example: Contents of buffer read

```
e ioa_ Type
$a
¢r¢f
¢b(test)
¢b(read)
```

Contents of buffer test

```
s/¬¢c.$//
d
e ioa_ "Give me a segment name."
w ¢r
q
```

Explanation of the read buffer

1. The first request in read is an escape to the command processor to call ioa_ to print the message "Type" on the user's terminal.

2. The second request ($a) places the editor in input mode to append text to the end of the current buffer (presumably buffer 0).

3. One line is read from the user's terminal (¢r) and the append request is terminated (¢f).

4. The contents of buffer test is executed (i.e., read "calls" the test "subroutine").

5. When and if test "returns", the contents of buffer read are executed again (i.e., read "calls" itself).

Explanation of the test buffer

1. The first line uses a substitute request to test the current line (i.e., the line just read in by the above append request) for the input terminating sequence (a line consisting only of "."). If the regular expression in the substitute request fails to find the terminating sequence, the remaining requests in buffer test are ignored (i.e., the test subroutine "returns" to its caller).

© Copyright, 1973, Massachusetts Institute of Technology
and Honeywell Information Systems Inc.

2. If the terminating sequence is found in step 1, the blank line that previously contained the terminating sequence is deleted.

3. Again ioa_ is used to type a message to the user. This time, the macro asks the user for a segment name under which the input lines appended by the read subroutine are to be stored.

4. The contents of the current buffer containing the input lines are written into a segment, the name of which is read from the terminal by the ¢r escape sequence.

5. The macro exits from the editor with a quit request. If the quit request were not included, qedx would expect further instructions from the user's terminal at this point.

8. Initialization of Macros

The editor provides a means through which a qedx macro can be initiated directly from command level. As indicated earlier, qedx can be invoked in the following fashion.

```
qedx path
```

The above command is equivalent to entering the editor with the simple command

```
qedx
```

and immediately executing the following series of requests.

```
b(exec)
r path.qedx
b0
¢b(exec)
```

This request sequence reads the initial macro segment into buffer exec, changes the current buffer back to buffer 0 and executes the contents of buffer exec. This series of requests is sufficient to allow a multi-buffer macro to be initialized. For example, the macro given in the previous example can be initialized and run from a segment with the following contents.

```
b(read)$a
e ioa_ Type
```

© Copyright, 1973, Massachusetts Institute of Technology
and Honeywell Information Systems Inc.

```
$a
¢c¢r¢c¢f
¢c¢b(text)
¢c¢b(read)
¢f
b(test)$a
s/¬¢c¢c.$//
d
e ioa_ "Give me a segment name."
w ¢c¢r
q
¢f
b0
¢b(read)
```

The contents of the buffers read and test are initialized with append requests. Notice that all escape sequences to be placed into a buffer as literal text must be preceded by a ¢c escape sequence. Thus, the second line to be input to buffer "read" is input as

```
¢c¢r¢c¢f
```

to produce the following line in the target buffer.

```
¢r¢f
```

In addition to the above, the qedx editor can be invoked with more than one argument. Thus,

```
qedx read path
```

is the equivalent of

```
qedx
b(exec)
r read.qedx
b(args)
a
path
¢f
b0
¢b(exec)
```

© Copyright, 1973, Massachusetts Institute of Technology
and Honeywell Information Systems Inc.

If the contents of read.qedx is:

```
r ¢b(args)
```

then the contents of buffer exec and buffer args become:

exec	args
r ¢b(args)	path

and the request ¢b(exec) causes the segment with the pathname path to be read into buffer 0. At that point the editor waits for further commands from the user.

With the same contents of read.qedx the invocation

```
qedx read path 1,$s/x/y/ w q
```

enters into the buffers exec and args the following:

exec	args
r ¢b(args)	path 1,$s/x/y/ w q

This causes the editor to read the segment path into buffer 0, substitute for every occurence of x the character y, write out the segment path and quit, returning to command level.

Notes

Since the name of the segment to be read in appears on the command line, this feature allows users to use abbreviations (see the write-up of the abbrev command) for the names of segments to be edited.

There is no safeguard to keep the editor from changing a buffer from which it is also accepting editor requests. If this is attempted, havoc may well be the result.

© Copyright, 1973, Massachusetts Institute of Technology
and Honeywell Information Systems Inc. (END)

Name: runoff, rf

The runoff command is used to type out text segments in manuscript form. Output lines are built from the left margin by adding text words until no more words will fit on the line; the line is then justified by inserting extra blanks to make an even right margin. Up to twenty lines of header and footer can be printed on each page. The pages can be numbered, lines can be centered, and equations can be formatted. Space can be allowed for diagrams. Detailed control over margins, spacing, headers, justification, numbering, and other aspects of format is provided by control lines beginning with a period interspersed within the text, but not appearing in the output segment. The output can be printed page by page to allow positioning of paper, or it can be directed into a segment. Characters not available on the device to which output is directed are replaced by blanks. If special symbols must later be hand drawn, a separate segment can be created which contains reminders as to where each symbol should be placed. The user may define variables and cause expressions to be evaluated; in combination with the ability to refer to (and sometimes modify) variables connected with the workings of the runoff command, the user has extensive control over the processing of his text.

Usage

runoff pathname1 ... pathname*n* -control_args-

1) pathname*i* is the pathname of an input segment or multi-segment file (MSF) named entryname.runoff; however, the .runoff suffix may be supplied in the entryname. If more than one pathname is specified, they are treated as if runoff had been invoked separately for each one. The segments are printed in the order in which they occur in the invocation of the command.

2) control_args may be chosen from the following list. Any control argument specified anywhere in the command invocation applies to all segments, and control arguments may be intermixed arbitrarily with segment names. Control arguments must be preceded by a minus sign.

© Copyright, 1972, Massachusetts Institute of Technology
All rights reserved.

-ball n, -bl n Output is converted to a form suitable for an n typeball on a unit equipped with a Selectric (R) typing element. Acceptable ball numbers are 041, 012, 015, and 963. The default is the form of the terminal device being used. Use of this control argument overrides any specification set by the -device control argument (below).

-character, -ch When this control argument is used, certain key characters in the output will by flagged by putting the line containing the key character in a segment named entryname.chars. The normal output is not affected. Page and line numbers referring to the normal output will appear with each flagged line, and reminder characters, enclosed by color-shift characters, will be substituted for the key characters. The default set of key and reminder characters corresponds to those unavailable with a 963 typeball, as follows:

left square bracket	<
right square bracket	>
left brace	(
right brace	)
tilde	t
grave accent	'

The key and reminder characters may be changed by use of the .ch control line; specifying a blank reminder character removes the associated key character from the set of key characters. If a key character would print normally in the output, it should also appear in a .tr control line to turn it into a blank in the output.

-device n, -dv n Output is prepared compatible with the device specified. This is usually used when the output is stored in a segment to be printed elsewhere. Suitable devices are consoles 2741, 1050, 37, and the bulk output printers, 202 or 300. Use of this control argument overrides any specification set by use of the -ball control argument; if both are used in one invocation of runoff, the last one encountered will prevail.

© Copyright, 1972, Massachusetts Institute of Technology
All rights reserved.

If neither -device nor -ball was specified, the default device type is that from which the user is logged in; any unrecognized type is assumed to support the entire ASCII character set.

-from n, -fm n
Printing starts at the page numbered n. If the -page control argument is used, printing starts at the renumbered page n.

-hyphenate, -hph
When this control argument is used, a procedure named hyphenate_word_, which the user supplies, will be invoked to perform hyphenation when the next word to be output will not fit in the space remaining in a line (see Hyphenation Procedure Calling Sequence near the end of this document). Otherwise, no attempt will be made to hyphenate words.

-indent n, -in n
When this control argument is used, the output will be indented n spaces from the left margin (default indentation is 0 except for "-device 202" (the default for -segment), which is 20; see also -number below). This space is in addition to whatever indentation is established by use of the .in control word.

-no_pagination,-npgn
Page breaks in the output are suppressed.

-number, -nb
Source line numbers will be printed in the left margin of the output; minimum indentation of 10 is forced.

-page n, -pg n
This control argument changes the initial page number to n. All subsequent pages are similarly renumbered. Note that if the control word .pa is used within the segment the -page control argument is overridden and the page is numbered according to the .pa control line.

-parameter arg, -pm arg
The argument arg is assigned as a string to the internal variable "Parameter".

© Copyright, 1972, Massachusetts Institute of Technology
All rights reserved.

-pass n — The source segments will be processed n times to permit proper evaluation of expressions containing symbols which are defined at a subsequent point in the input. No output is produced until the last pass.

-segment, -sm — When this control argument is used, the output will be directed to the segment or MSF entryname.runout. This control argument assumes by default that the material is to be dprinted, so the segment is prepared compatible with device 202 unless another device is specified; thus, unless overridden by the -indent control argument, each printed line in the output segment is preceded by 20 leading spaces so that the text will be approximately centered on the page when dprinted.

-stop, -sp — The runoff command will wait for a carriage return from the user before beginning typing and after each page of output.

-to n — Printing ends after the page numbered n.

-wait, -wt — The runoff command will wait for a carriage return from the user before starting output, but not between pages.

Notes

A runoff input segment contains two types of lines: control lines and text lines. A control line begins with a period; all other lines are considered text lines. A two-character control word appears in the second and third character positions of each control line. The control word may take a parameter which is separated from the control word by one or more spaces. Lines which are entirely blank are treated as if they contained a ".sp 1" control line.

Text lines contain the material to be printed. If an input line is too short or too long to fill an output line, material is taken from or deferred to the next text line. A line beginning with a space is interpreted as a break in the text (e.g. the beginning of a new paragraph) and the previous line is printed as is.

Tab characters (ASCII HT) encountered in the input stream are converted to the number of spaces required to get to the next tab position (11, 21, ...).

© Copyright, 1972, Massachusetts Institute of Technology
All rights reserved.

When an input text line ends with any of the characters ".", "?", "!", ";", or ":", or with ".", "?", or "!" followed by a double quote or ")", two blanks will precede the following word (if it is placed on the same output line), instead of the normal single blank.

The maximum number of characters per input or output line is 361; this permits 120 underlined characters plus the NL character.

Terminology

Two separate concepts are relevant to understanding how runoff formats output: fill mode and adjust mode. In fill mode, text is moved from line to line when the input either exceeds or cannot fill an output line. Adjust mode right justifies the text by inserting extra spaces in the output line, with successive lines being padded alternately from the right and from the left. Note that initial spaces on a line are not subject to adjustment. Fill mode can be used without adjust, but in order for adjust to work, fill mode must be in effect.

The line length is the maximum number of print positions in an output line, including all spaces and indentations, but not including margins set or implied by the -device, -indent, or -number control arguments.

A break insures that the text that follows will not be run together with the text before the break. The previous line is printed out as is, without padding.

Vertical spacing within the body of the text is controlled by the three commands .ss, .ds, and .ms. Single spacing, which is the default, is set by .ss, double spacing is set by .ds, and ".ms n" is used for multiple spacing. There are (n-1) blank lines between text lines for .ms.

A page eject insures that no text after the control line will be printed on the current page. The current page is finished with only footers and footnotes at the bottom, and the next text line begins the following page.

© Copyright, 1972, Massachusetts Institute of Technology
All rights reserved.

There are four margins on the page vertically. The first margin on the page is the number of blank lines above the first header, and is set by the .m1 control line. The second, set by .m2, concerns the number of lines between the last header and the first line of text. The third is between the last line of text and the first footer, set by .m3. The fourth is below the last footer, set by .m4. The default for the first and fourth margins is four lines; for the second and third, the default is two lines.

As the output is being prepared, a page number counter is kept. This counter can be incremented or set by the user. The current value of the counter can be used in a header or footer through the use of the symbol "%". A page is called odd (even) if the current value of the counter is an odd (even) number.

A header is a line printed at the top of each page. A footer is a line printed at the bottom of each page. A page may have up to twenty headers and twenty footers. Headers are numbered from the top down, footers from the bottom up. The two groups are completely independent of each other. Provision is made for different headers and footers for odd and even numbered pages. Both odd and even headers (footers) can be set together by using .he (.fo). They are set separately by using .eh, .oh, .ef, and .of.

A header/footer definition control line has two arguments, the line number (denoted in the control line descriptions as "#"), and the title.

The line number parameter of the control line determines which header or footer line is being set. If the number is omitted, it is assumed to be 1, and all previously defined headers or footers of the type specified (odd or even) are cancelled. Once set, a line is printed on each page until reset or cancelled.

The title part of the control line begins at the first non-blank character after the line number. This character is taken to be the delimiting character, and may be any character not used in the rest of the title. If the delimiting character appears less than four times, the missing last parts of the title are taken to be blank. The three parts of the title are printed left justified, centered, and right justified, respectively, on the line. Any or all parts of the title may be null. Justification and centering of a header or footer line are derived from the line length and identation in effect at the time of the definition of the header or footer, and will be used whenever that line is output, regardless of the values at the time of use.

© Copyright, 1972, Massachusetts Institute of Technology
All rights reserved.

Omitting the title in the control line cancels the header or footer with that number, including its space on the page. A blank line in the header or footer may be achieved by a title consisting entirely of four delimiting characters. Omitting both number and title cancels all headers or footers of the type specified.

Expressions and Expression Evaluation

An expression may be either arithmetic or string, and consists of numbers and operators in appropriate combinations. All operations are performed in integer format, except that string comparisons are performed on the full lengths of the strings.

Operators in order of precedence:

¬ (bit-wise negation), - (unary)
*, /, ¢ (remainder)
+, - (binary)
=, <, >, ≠, ≤, ≥
(comparison operators, yield -1 (true) or 0 (false))
& (bit-wise AND)
| (bit-wise OR), ≡ (bit-wise equivalence)

Parentheses may be used for grouping.

Blanks are ignored outside of constants.

Octal numbers consist of "#" followed by a sequence of octal digits.

String constants are surrounded by the double quote character; certain special characters are defined by multiple character sequences beginning with the character *, as follows:

** yields	asterisk-character
*"	double-quote-character
*b	backspace character
*n	new-line character
*t	horizontal tab character
*s	space character
*cnnn	character whose decimal value is nnn (1 to 3 digits)

Concatenation of strings is performed by the juxtaposition of the strings involved, in order, left to right.

© Copyright, 1972, Massachusetts Institute of Technology
All rights reserved.

For positive i, k,

string_expression(i)

and

string_expression(i, k)

are equivalent to the PL/I substr built-in function references

substr(string_expression, i)

and

substr(string_expression, i, k)

respectively.

For negative i, the substring is defined as starting -i characters from the rightmost end of the string; for negative k, the substring ends -k characters from the end of the string.

Evaluation of substrings takes place after any indicated concatenations; string operations have higher precedence than all the binary operations.

In any context other than a ".sr" control line or in a string comparison, a string expression is converted to an integer in such a way that a one-character string results in the ASCII numeric value of the character.

Expression evaluation takes place under the following conditions:

1) In .sr and .ts control lines;
2) In all control lines which accept an "n" or "±n" argument.

Definition and Substitution of Variables

Variables may be defined by the use of the .sr control line; their values may be retrieved thereafter by a symbolic reference. Names of the variables are composed of the upper- and lower-case alphabetic characters, decimal digits, and "_", with a maximum length of 361 characters. When a variable is defined, it is given a type based on the type of the expression which is to be its value, either arithmetic or string. Variables which are undefined at the time of reference yield the null string, which is equivalent to an arithmetic 0.

© Copyright, 1972, Massachusetts Institute of Technology
All rights reserved.

In substitution of variables, the name of the variable is enclosed by "%"; other occurrences of the character "%" encountered during substitution of variables are replaced by the value of the page counter; if a "%" character is to occur in the resulting output, it must be coded as "%%" (but see also .cc).

Substitution of variables may occur

1) In expressions if a "%" is found as either the first or second character following the spacing after the control word (substitution of variables takes place before expression evaluation);
2) In .ur control lines;
3) In all titles ('part1'part2'part3'), whether in header/footer control lines, or as equation lines.

Many of the variables internal to runoff are available to the user (a complete list will be found at the end of this document); these include control argument values (or their defaults), values of switches and counters, and certain special functions. However, the user need not worry about naming conflicts, since an attempt to re-define an internal variable that is not explicitly modifiable will merely make it inaccessible to the user, but will cause no harm to the operation of the command.

Two special builtin symbols in runoff are provided for use in footnote and equation numbering: "Foot" contains the value of the next footnote number available (or the current footnote if referred to from within the text of the footnote), and "Eqcnt" is provided for equation numbering. The value of "Foot" is incremented by one when the closing .ft of a footnote is encountered. Any reference to "Eqcnt" provides the current value, and causes its value to be incremented by one automatically; thus its value should be assigned to a variable, and the variable should then be used in all further references to that equation number.

© Copyright, 1972, Massachusetts Institute of Technology
All rights reserved.

Default Conditions

When no control words are given, runoff prints the text single spaced, right adjusted, with no headers, no footers, and no page numbers.

If page numbers are substituted in headers or equations, they will be arabic.

A page consists of 66 lines, numbered 1 through 66. The first line is printed on line 7, and the last on line 60, if no headers or footers are used. If headers are used, there will be four lines of top margin, the headers, two blank lines, and then the text. If footers are used, there will be two lines skipped after the text, footers printed, and four lines of bottom margin.

A line is 65 characters long; the left margin is that of the typewriter. The output is compatible with whatever is normal for the device from which the runoff command is executed. The entire segment is printed, with no wait before beginning or between pages.

© Copyright, 1972, Massachusetts Institute of Technology
All rights reserved.

Control Line Formats

This section gives a description of each of the control words which may be interspersed with the text for format control. Control lines do not cause an automatic break unless otherwise specified. Arguments of the control words are in the following form:

#	integer constant
n	integer expression
±n	integer expression preceded by optional + or -
<expression>	arbitrary expression (string or integer)
c	character
cd	character pair
f	segment name
'part1'part2'part3'	a title whose parts are to be left justified, centered, and right justified respectively.

(blank line)
A blank line occurring in the text is treated as if it were a ".sp 1" control line.

.ad Adjust: text is printed right justified. Fill mode must be in effect for right justification to occur. Fill mode and adjust mode are the default conditions. This control line causes a break.

.ar Arabic numerals: when numeric variables are substituted into text or control lines as a result of a .ur control line, or into a title or equation as it is printed, they are in arabic notation. This is the default condition.

.bp Begin page: the next line of text begins on a new page of output. This control line causes a break.

.br Break: the current output line is finished as is, and the next text line begins on a new output line.

.cc c Control character: this control line changes the character used to surround the names of symbolic variables when they are referenced to c. The default special character is "%". The character specified by c must thereafter be used to refer to symbolic variables, while percent signs are treated literally. ".cc %" or .cc restores the percent sign as the special character.

© Copyright, 1972, Massachusetts Institute of Technology
All rights reserved.

.ce n Center: the next n text lines are centered. If n is missing, 1 is assumed. This control line implies ".ne n" (or ".ne 2n" if doublespacing) so that all lines centered will be on the same page. A break occurs.

.ch cd.. Characters: each occurrence of the character c will be replaced in the .chars segment by the character d, set off by color-shift characters. If the d character is blank, or an unpaired c character appears at the end of the line, the c character will not be flagged, and will occur as itself in the .chars segment, or not at all if no other character on the line was flagged.

.ds Double space: begin double spacing the text. This control line causes a break.

.ef # 'part1'part2'part3'
  Even footer: this defines even page footer line number #. See the section entitled Terminology.

.eh # 'part1'part2'part3'
  Even header: this defines even page header line number #. See the section entitled Terminology.

.eq n Equation: the next n text lines are taken to be equations. If n is missing, 1 is assumed. This control line implies ".ne n" (or ".ne 2n" if doublespacing) so that all equations will be on the same page. The format of the equations should be 'part1'part2'part3' just as in headers.

.ex text Execute: the remainder of the control line (text) is passed to the Multics command processor. Substitution of variables may occur if the first or second character of text is "%".

.fh 'part1'part2'part3'
  Footnote header: before footnotes are printed, a demarcation line is printed to separate them from the text. The format of this line may be specified through the title in the .fh control line. This title is printed in the same manner as headers. The default footnote header is a line of underscores from column one to the right margin.

© Copyright, 1972, Massachusetts Institute of Technology
All rights reserved.

.fi — Fill: this control line sets the fill mode. In fill mode, text is moved from line to line to even the right margin, but blanks are not padded to justify exactly. This is the default condition. Since right justification is also the default condition, getting a slightly even right margin without adjustment is accomplished by use of the .na control line. This control line causes a break.

.fo # 'part1'part2'part3'
Footer: even and odd footers are set at the same time; this is equivalent to
.ef # 'part1'part2'part3'
.of # 'part1'part2'part3'
See the section entitled Terminology.

.fr c — Footnote reset: this control line controls footnote numbering according to the argument c. Permitted values of this argument are:
- t Footnote counter is reset at the top of each page. This is the default condition.
- f Footnote counter runs continuously through the text.
- u Suppresses numbering on the next footnote.

.ft — Footnote: when .ft is encountered all subsequent text until the next .ft line is treated as a footnote. Any further text on the .ft line will be ignored. If a footnote occurring near the bottom of a page will not fit on the page, as much as necessary will be continued at the bottom of the next page. If a footnote reference occurs in the bottom or next to bottom line of a page, the current page will be terminated and the offending line printed at the top of the succeeding page.

.gb xxx — Go back: the current input segment is searched from the beginning until a line of the form ".la xxx" is found; "xxx" in this case means "the rest of the line". Processing is continued from that point.

.gf xxx — Go forward: same as .gb, except search forward from the current position in the input segment.

© Copyright, 1972, Massachusetts Institute of Technology
All rights reserved.

.he # 'part1'part2'part3'

Header: even and odd headers are set at the same time. This is equivalent to

.eh # 'part1'part2'part3'

.oh # 'part1'part2'part3'

See the section entitled Terminology.

.if f <expression>

Insert file: the segment with pathname f.runoff is inserted into the text at the point of the ".if f" request. The inserted segment may contain both text and control lines. No break occurs. The effect is as if the control line were replaced by the segment. Inserts may be nested to a maximum depth of 30. If a second argument is provided, it will be evaluated in the same fashion as the expression in .sr, and its value and type will be associated with the identifier "Parameter"; otherwise the value of "Parameter" remains unchanged (or undefined) (prior values of "Parameter" are not pushed down).

.in ±n Indent: the left margin is indented n spaces by padding n leading spaces on each line. The right margin remains unchanged. By default n is 0. The margin can be reset with another ".in n" request. Either .in or ".in 0" resets the original margin. If n is preceded by a plus or a minus sign, the indentation is changed by n rather than reset. This control line causes a break.

.la xxx Label: defines the label xxx for use as the target of the .gb or .gf control word.

.li n Literal: this request causes the next n lines to be treated as text, even if they begin with ".". If n is not given 1 is assumed.

.ll ±n Line length: the line length is set to n. The left margin stays the same, and no break occurs. The default for n is 65 both initially and if n is omitted in the .ll control line. If n is preceded by a plus or a minus sign, the line length is changed by n rather than reset.

© Copyright, 1972, Massachusetts Institute of Technology
All rights reserved.

.ma ±n — Margins: top and bottom margins are set to n lines. If n is preceded by a plus or a minus sign, the margin is changed by n rather than reset. The margin is the number of lines printed above the first header and below the last footer. The default is four lines. This control line is equivalent to

```
.m1 ±n
.m4 ±n
```

.mp ±n — Multiple pages: format the output text so that it prints on every nth page. This control line is valid only for output intended for the bulk printer. The default value is 1.

.ms ±n — Multiple space: begin multiple spacing text, leaving (n-1) blank lines between text lines. If n is preceded by a plus or a minus sign, the spacing is changed by n rather than reset. If n is not given, 1 is assumed. This control line causes a break.

.m1 ±n — Margin 1: the margin above the first header is set to n lines, or changed by n if n is signed. The default is four lines.

.m2 ±n — Margin 2: the number of blank lines printed after the last header and before the first line of text is set to n, or changed by n if n is signed. The default is two lines.

.m3 ±n — Margin 3: the number of blank lines printed after the last line of text and before the first footer is set to n, or changed by n if n is signed. The default is two lines.

.m4 ±n — Margin 4: the margin below the last footer is set to n lines, or changed by n if n is signed. The default is four lines.

.na — No adjust: the right margin is not adjusted. This does not affect fill mode; text is still moved from one line to another. This control line causes a break.

.ne n — Need: a block of n lines is needed. If n or more lines remain on the current page, text continues as before; otherwise, the current page is ejected and text continued on the next page. No break is implied. The default value is 1.

© Copyright, 1972, Massachusetts Institute of Technology
All rights reserved.

.nf — No fill: fill mode is suppressed, so that a break is caused after each text line. Text is printed exactly as it is in the input segment. This control line causes a break.

.of # 'part1'part2'part3'
Odd footer: this defines odd page footer line number #. See the section entitled Terminology.

.oh # 'part1'part2'part3'
Odd header: this defines odd page header line number #. See the section entitled Terminology.

.op — Odd page: the next page number is forced to be odd by adding 1 to the page number counter if necessary. A break is caused and the current page is ejected.

.pa ±n — Page: the current line is finished as is (ie a break occurs) and the current page is ejected. The page number counter is set to n, or is changed by n if n was signed.

.pi n — Picture: if n lines remain on the present page, then n lines are spaced over; otherwise, the text continues as before until the bottom of the page is reached, then n lines are skipped on the next page before any text is printed. Headers are printed normally, and the space is below the headers. This option may be used to allow for pictures and diagrams. If several .pi control lines are used, each n is added to the number of lines pending and the total is checked against the space remaining on the page. All pending space is allotted together. If the total is greater than the usable space on a page, the next page contains only headers and footers and the rest of the space is left on the following page.

.pl ±n — Page length: the page length is set to n lines. The default is 66 lines. If n is preceded by a plus or a minus sign, the page length is changed by n rather than reset.

.rd — Read: one line of input is read from the stream "user_input"; this input line is then processed as if it had been encountered instead of the .rd control line. Thus it may be either a text line or a control line; a break occurs only if the replacement line is a control line which causes a break, or a text line beginning with one or more spaces, or a blank line.

© Copyright, 1972, Massachusetts Institute of Technology
All rights reserved.

.ro — Roman numerals: numeric-to-string conversions required by an explicit .ur control line (q.v.) or in titles or equations will result in roman numerals in the evaluated text. This may be reset to arabic numerals (the default) by use of the .ar control line.

.rt — Return: cease processing characters from the current input segment.

.sk n — Skip: n page numbers are skipped before the next new page by adding n to the current page number counter. No break in text occurs. This control argument may be used to leave out a page number for a figure. If n is not given 1 is assumed.

.sp n — Space: space n lines. If n is not given, 1 is assumed. If not enough lines remain on the current page, footers are printed and the page ejected, but the remaining space is not carried over to the next page. This control line causes a break.

.sr name <expression>
Set reference: associate the value of <expression> with the identifier name. The type of name will be set to the type of <expression> (either numeric or string); if the expression is not provided, or cannot be properly evaluated, a diagnostic message will be printed. name may be either a user-defined identifier or one of the built-in symbols which the user may set. (see Built-in Symbols below.)

.ss — Single space: begin single spacing text. This is the default condition. This control line causes a break.

.tr cd.. — Translate: the nonblank character c is translated to d in the output. An arbitrary number of cd pairs can follow the initial pair on the same line without intervening spaces. An unpaired c character at the end of a line will translate to a blank character. (Translation of a graphic character to a blank only in the output is useful for preserving the identity of a particular string of characters, so that the string will not be split across a line, nor have padding inserted within it.)

.ts n — Test: process the next input line if the value of n does not equal zero (false). The default value is 1.

© Copyright, 1972, Massachusetts Institute of Technology
All rights reserved.

.ty xxx — Type: write xxx (ie the rest of the control line) onto the stream "error_output". Substitution of variables may occur if the first or second character of xxx is "%".

.un n — Undent: the next output line is indented n spaces less than the current indentation. Adjustment, if in effect, will occur only on that part of the line between the normal left indentation and the right margin. If n is not specified, its value is the current indentation value (ie, the next output line will begin at the current left margin). This control line causes a break.

.ur text — Use reference: the remainder of the .ur control line (text) will be scanned, with variables of the form "%name%" replaced by their corresponding values (converted back to character string form if they were numeric). The line thus constructed is then processed as if it had been encountered in the original input stream (e.g., it may be another control line, including possibly another .ur).

.wt — Wait: read one line from the stream "user_input", and discard it (cf .rd).

.* — This line is treated as a comment and ignored. No break occurs.

. — This line is treated as a comment and ignored with respect to the output segment. However, the line is printed in the appropriate place in the .chars output segment.

© Copyright, 1972, Massachusetts Institute of Technology
All rights reserved.

Summary of Control Arguments and Control Line Formats

Control arguments

-ball n, -bl n
: Convert output to a form suitable for an n typeball.

-character, -ch
: Create entryname.chars, listing page and line numbers with red reminder characters where certain characters, normally not printable, must be drawn in by hand.

-device n, -dv n
: Prepare output compatible with device n.

-from n, -fm n
: Start printing at the page numbered n.

-hyphenate, -hph
: Call user-supplied procedure to perform hyphenation.

-indent n, -in n
: Set initial indentation to n.

-no_pagination, -npgn
: Suppress page breaks.

-number, -nb
: Print source segment line numbers in output.

-page n, -pg n
: Change the initial page number to n.

-parameter arg, -pm arg
: Assign arg as a string to the internal variable "Parameter".

-pass n
: Make n passes over the input.

-segment, -sm
: Direct output to the segment or MSF "entryname.runout", where entryname is the name of the input segment.

-stop, -sp
: Wait for a carriage return before each page.

-to n
: Finish printing after the page numbered n.

-wait, -wt
: Wait for a carriage return before the first page.

© Copyright, 1972, Massachusetts Institute of Technology
All rights reserved.

Control line formats

The following conventions are used to specify arguments of control words:

c	character
cd	character pair
exp	expression (either numeric or string)
#	integer constant
n	integer expression
±n	± indicates update by n; if sign not present set to n
f	segment name
t	title of the form 'part1'part2'part3'

Request	Break	Default	Meaning
(blank line)	yes		Equivalent to ".sp 1"
.ad	yes	on	Right justify text
.ar	no	arabic	Arabic page numbers
.bp	yes		Begin new page
.br	yes		Break, begin new line
.cc c	no	%	Change special character from % to c
.ce n	yes	n=1	Center next n lines
.ch cd....	no		Note "c" in .chars file as "d"
.ds	yes	off	Double space
.ef # t	no		Defines even footer line #
.eh # t	no		Defines even header line #
.eq n	yes	n=1	Next n lines are equations
.ex xxx	no		Call command processor with "xxx"
.fh t	no	line of underscores	Format of footnote demarcation line
.fi	yes	on	Fill output lines
.fo # t	no		Equivalent to: .ef # t .of # t
.fr c	no	t	Controls footnote numbering: "t" reset each page; "f" continuous; "u" numbering suppressed for next footnote.
.ft	no		Delimits footnotes
.gb xxx	no		"go back" to label xxx
.gf xxx	no		"go forward" to label xxx
.he # t	no		Equivalent to: .eh # t .oh # t
.if f exp	no		Segment f.runoff inserted at point of request; value of "exp" assigned to "Parameter"
.in ±n	yes	n=0	Indent left margin n spaces
.la xxx	no		Define label xxx

© Copyright, 1972, Massachusetts Institute of Technology
All rights reserved.

Request	Break	Default	Meaning
.li n	no	n=1	Next n lines treated as text
.ll ±n	no	n=65	Set line length to n
.ma ±n	no	n=4	Top and bottom margins set to n
.mp ±n	no	n=1	Print only every n-th page
.ms ±n	yes	n=1	Multiple space of n lines
.m1 ±n	no	n=4	Margin above headers set to n
.m2 ±n	no	n=2	Margin between headers and text set to n
.m3 ±n	no	n=2	Margin between text and footers set to n
.m4 ±n	no	n=4	Margin below footers set to n
.na	yes	off	Do not right justify
.ne n	no	n=1	Need n lines; begin new page if not enough remain
.nf	yes	off	Nofill; break after each input line
.of # t	no		Defines odd footer line #
.oh # t	no		Defines odd header line #
.op	yes		Next page number is odd
.pa ±n	yes		Begin page n
.pi n	no	n=1	Skip n lines if n remain; otherwise skip n on next page before any text
.pl ±n	no	n=66	Page length is n
.rd	no		Read one line of text from "user_input" and process it in place of .rd line
.ro	no	arabic	Roman numeral page numbers
.rt	no		"return" from this input segment
.sk n	no	n=1	Skip n page numbers before next new page
.sp n	yes	n=1	Space n lines
.sr sym exp	no		Assign value of "exp" to variable named "sym"
.ss	yes	on	Single space
.tr cd....	no		Translate nonblank character c into d on output
.ts n	no	n=1	Process the next input line only if n is non-zero
.ty xxx	no		Write "xxx" onto the stream "error_output"
.un n	yes	left margin	Indent next text line n spaces less
.ur text	no		Substitute values of variables in "text", and re-scan the line.

© Copyright, 1972, Massachusetts Institute of Technology
All rights reserved.

Request	Break	Default	Meaning
.wt	no		Read one line of text from "user_input" and discard it (for synchronization with console)
.*	no		Comment line; ignored
.	no		Comment line; ignored, but included in .chars output

© Copyright, 1972, Massachusetts Institute of Technology
All rights reserved.

Built-in Symbols

Only those symbols marked yes in the Set column may have values assigned by the user.

All symbols are of type Number unless they are specified to be of type string.

Control words and control arguments which affect the values of the variables are indicated in parentheses: (x/y) indicates that x sets the switch to true (-1), and y sets it false (0); (a) or (a, b, c) indicates that it is affected by a or by a, b and c.

Symbol	Set	Value
Ad		Adjust (.ad/.na)
Ce		Number of lines remaining to be centered (.ce)
CharsTable	yes	Translation table for .chars segment output (String) (.ch)
Charsw	yes	".chars" file is being created (-character)
ConvTable	yes	Translation table for output. Product of DeviceTable and TrTable (String) (.tr, -device)
Date		Date of this invocation of runoff; format is mm/dd/yy (String)
Device	yes	Type of device output is to be formatted for (-device, -ball, -segment)
DeviceTable	yes	Translation table for physical device (String) (-device)
Eq		Equation line counter (.eq)
Eqcnt	yes	Equation reference counter (incremented each reference)
ExtraMargin	yes	Indent entire text this many spaces (-segment, -device, -indent)
Fi		Fill switch (.fi/.nf)
FileName		Name of current primary input segment (String)
Filesw		True if output is going to a segment (-segment)
Foot	yes	Footnote counter (.ft, .fr)
FootRef	yes	Footnote reference string in footnote body (String)
Fp	yes	First page to print (set at the beginning of each pass to the value of From)

© Copyright, 1972, Massachusetts Institute of Technology
All rights reserved.

Symbol	Set	Value
Fr		Footnote counter reset switch
From	yes	Control argument value (-from)
Ft		Footnote processing switch (.ft)
Hyphenating	yes	True if an attempt to break a word should be made (-hyphenate)
In		Indent to here (.in)
InputFileName		Name of current input segment (String) (.if)
InputLines		Current line number in current source file
LinesLeft		Number of usable text lines left on this page
Ll		Line length (.ll)
Lp	yes	Last page to print (initialized each pass from To)
Ma1		Space above header (.ma, .m1)
Ma2		Space below header (.m2)
Ma3		Space above foot (.m3)
Ma4		Space below foot (.ma, .m4)
Ms		Spacing between lines (ss = 1, ds = 2, etc.) (.ms, .ss, .ds)
MultiplePagecount		Form feeds between pages to printer (.mp)
NestingDepth		Index into stack of input files (.if)
Nl		Last used line number
NNp	yes	Next page number (-page, .pa)
NoFtNo		True to suppress number on next footnote reference (.fr)
NoPaging	yes	True if no pagination is desired (-no_pagination)
Np	yes	Current page number (.pa, -page, initialized each pass from Start)
PadLeft		Alternate left/right padding switch (.un, .ad)
Parameter	yes	Argument passed during insert processing (-parameter, .if)
Passes	yes	Number of passes left to make (= 1 when printing is being performed) (-pass)
Pi		Space needed for pictures (.pi)
Pl		Page length (.pl)
Print	yes	Whether or not to print ((Fp $\leq$ Np $\leq$ Lp) & (Passes $\leq$ 1))
Printersw		Output is intended for bulk printer (-device, -segment)
PrintLineNumbers	yes	True if source line numbers are to be printed in output (-number)

© Copyright, 1972, Massachusetts Institute of Technology
All rights reserved.

Symbol	Set	Value
Roman		Roman numeral pagination (.ro/.ar)
Selsw		True if typeball other than 963 is being used (-ball)
Start	yes	Initial page number (-page, -start)
Stopsw	yes	Stop between pages of output (-stop)
TextRef	yes	Footnote reference string in main text (String)
Time		Local time, in seconds, since January 1, 1901.
To	yes	Last page to be printed (-to)
TrTable	yes	Translation table for user-supplied substitutions (String) (.tr)
Un		Undent to here (.un)
Waitsw	yes	Wait for input before printing first page (-wait)

© Copyright, 1972, Massachusetts Institute of Technology
All rights reserved.

Hyphenation Procedure Calling Sequence

The runoff command provides a means whereby a user-supplied program may be called whenever the space available on a line is less than the length of the next word (including attached punctuation, if any). The mechanism is activated by use of the -hyphenate control argument, and the PL/I calling sequence is provided below.

```
declare hyphenate_word_ entry(char(*) unaligned, fixed bin,
          fixed bin);

call hyphenate_word_(string, space, break);
```

1) string — is the text word which is to be split. (Input)

2) space — is the number of print positions remaining in the line. (Input)

3) break — is the number of characters from the word that should be placed on the current line; it should be at least one less than the value of space (to allow for the hyphen), and may be 0 to specify that the word is not to be broken. Thus if the word "calling" is to be split, and 6 spaces remain in the line, the procedure should return the value 4 (adjustment is performed after hyphenation). (Output)

© Copyright, 1972, Massachusetts Institute of Technology
All rights reserved.

(END)*